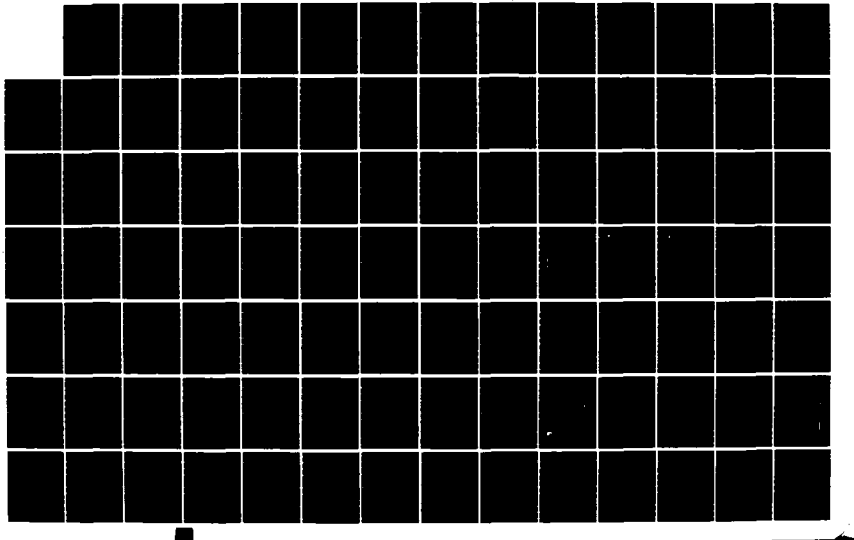
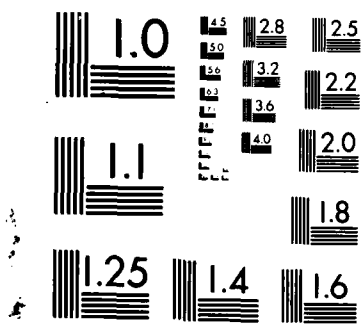
AD-A174 053 AAAI (AMERICAN ASSOCIATION ON ARTIFICIAL INTELLIGENCE) 1/2
WORKSHOP ON AI (AR..(U) ARMY CONCEPTS ANALYSIS AGENCY
BETHESDA MD A GERSTENFELD ET AL. AUG 86
UNCLASSIFIED F/G 9/2 NL





AAAI Workshop

on

AI and Simulation

August 11, 1986

University of Pennsylvania
Philadelphia, PA

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A174053 — 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) American Association on Artificial Intelligence Workshop on Artificial Intelligence and Simulation	5. TYPE OF REPORT & PERIOD COVERED Workshop Report 11 Aug 86 — 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Arthur Gerstenfeld, Worcester Polytecnic Richard Modjeski, US Army Concepts Analysis Agency Y.V. (Ramana) Reddy, West VA University Marilyn Stelzner, IntelliCorp	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Concepts Analysis Agency 8120 Woodmont Ave, Bethesda, Md 20814-2797	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Concepts Analysis Agency 8120 Woodmont Ave, Bethesda, Md 20814-2797	12. REPORT DATE August 1986 — 13. NUMBER OF PAGES 172
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified — 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This workshop was sponsored and funded by the American Association on Artificial Intelligence.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Science (U)
Artificial Intelligence (U)
Computer Simulation Modeling (U)
Expert Systems (U)
Knowledge Engineering (U)
Modeling (U)
Operations Research (U)
Qualitative Simulation (U)
Simulation (U)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The first Artificial Intelligence (AI) and simulation workshop was held during the National Conference on Artificial Intelligence on August 11, 1986 at the University of Pennsylvania (Wharton Hall). The following abstracts and papers were presented at the workshop.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

NOT USED

Workshop Organizing Committee

Dr. Arthur Gerstenfeld
Worcester Polytechnic Institute
Department of Management
Worcester, MA 01609

Dr. Richard Modjeski
U.S. Army Concepts Analysis Agency
8120 Woodmont Ave.
Bethesda, MY 20814-2797

Dr. Y. V. (Ramana) Reddy
Department of Statistics & Computer Science
West Virginia University
Morgantown, WV 26506

Marilyn Stelzner
IntelliCorp
1975 El Camino Real West
Mountain View, CA 94040

A1

Table of Contents
(by Abstract Number)

[23] "Reasoning About Diagnosis and Treatment in a Causal Time-Varying Domain Using Semi-Quantitative Simulation and Inference"
Lawrence E. Widman, M.D., Ph.D.

[27] "Rand Strategy Assessment System"
Steven C. Bankes

[28] "Orthogonal Views of a Simulation"
Dr. Malcolm R. Railey

[30] "A Knowledge-Based Simulation Model for Evaluating Alternative Courses of Action"
Ina Ghaznavi

[31] "SIMSMART: Dynamic Simulation for Engineering Design, Operator Training and Automated Control of Industrial Processes"
Don Waye

[32] "Abstract for Engineering Workshop AAAI 86"
R. M. Adler

[33] "AI Research at Martin Marietta Baltimore Aerospace"
Martin J. Dudziak

[36] "Integrating Truth Maintenance and Knowledge Based Simulation"
Philip McBride

[40] "An Autonomous Vehicle Workstation"
Dan Donahue

[42] "Simulation-Based Robot Cell Diagnosis"
C.N. Lee, Ph.D.

[43] "Device Modeling and Simulation in an Automatic Programming System"
Sol J. Greenspan

[46] "Research Activities in AI and Simulation"
Bernard P. Zeigler

[47] "Current Research in Qualitative Simulation"
Peter D. Karp

[50] "Research in Qualitative Simulation"
Stephen Zvolner

[51] "RiTSE: The Reactor Trip Simulation Environment"
Steven A. Epstein

[52] "Qualitative Simulation of Chemical Process Systems"
Mark A. Kramer

[53] "Petri Net Simulation of Rule Based Systems"
Allen Rude

[54] "Model Identification Using Heuristic Search"
S. Hanagud, Professor

[55] "Using AI-Based Simulation to Model Manufacturing Facilities"
Perry A. Zalevsky

[56] "Reasoning about Signals in an Expert System for Simulator Engineers"
Robert Prager

[58] "Knowledge-Based Simulation"
Jeff Rothenberg

[59] "Wrapping AI Techniques around Simulation and other Operations Research tools to help Manufacturing Managers"
Richard Berenson

[60] "Application of the Butterfly Parallel Processor to the Simulation and Control of Complex Systems"
David M. Fram

[61] "Abstract of work in AI and Simulation"
Kjell Tangen

[62] "The Analogical Representation of Liquids in Naive Physics"
Prof. Francesco Gardin

[64] "Knowledge Based Simulation Environments for Dynamic Systems"
Pertti Lounamaa

QUALITATIVE SIMULATION:
A FOCUS ON AIR TRAFFIC CONTROL

A. Gerstenfeld*
Y. Pan**

ABSTRACT

Conventional mathematical algorithms for aiding air traffic control have been suggested by previous reasearchers [1,2,]. However, the use of purely numerical representations and manipulations obscures the necessary "elegance" and heuristics needed in air traffic control. In fact it has been stated that no two controllers would handle a problem in the same way.

As an alternative to conventional algorithmic methods, other previous research has investigated the use of Knowledge-Based System technology [3,4,5,]. Most knowledge-based systems are usually developed in LISP,

*A. Gerstenfeld received his Ph.D. from MIT and currently holds an Endowed Chair at Worcester Polytechnic Institute. He has written four books and more than 30 articles which have appeared in academic journals.

**Y. Pan is currently a Senior Research Scientist at Worcester Polytechnic Institute. He was formerly with the Laboratory of Information and Decision Systems, MIT in 1981-1983 and Director of Automation with Shenyang Institute of Automation, Chinese Academy of Sciences in 1962-1980; 1983-1985.

however, the speed at which LISP machines can solve problems is often not sufficient. One solution to this problem is to optimize the LISP code for execution speed by using conventional algorithm techniques for parts of the system. This can be done in "C" and through a LISP/C interface achieve "the best of all possible worlds."

As pointed out at the 1986 AAAI conference [6] traditional algorithmic computer programs based on detailed mathematical models, yield the most accurate results but require large amounts of process data and long computational times. Furthermore, quantative predictions can be undersirable as insightful concepts are qualitative in nature. Human experts are able to make accurate qualitative predictions of the consequences of their control actions. A basic challenge in all qualitative simulation methods is to reduce the ambiguities.

Part of our approach is to use a frame hierarchy. Minsky points out, [7] that this method allows the programmer to associate methods with objects which are inherited. It is not surprising that languages are being developed to express both the dynamic knowledge of discrete events and the declarative knowledge of AI frame paradigms [8].

The initial training for air traffic control takes

place at the FAA Academy in Oklahoma City. Following that, approximately four years is spent in training at a control center. The training is a combination of on-the-job and supervised practice on a dynamic simulator (DYSIM). The later form of learning involves understanding the advise of an expert. As one observes the latest developments in the areas of simulation methodology, simulation language, computer graphics, computer hardware, and expert system research, it is impossible to escape the conclusion that AI-based expert simulations will soon be available to perform the tasks we previously believed could only be done by humans. The direction of innovations are changing and as described by Gerstenfeld in several previous articles, future successful innovations will be systems innovations [9,10,11,].

The system we describe in this paper was written in LISP and built on an IBM/AT with extended memory and a high resolution color monitor. We originally figured on using an artificial intelligence shell, but were not able to obtain the results we wanted in that way. We, therefore, reached the conclusion that it would be best to write the program in LISP for the intelligence portion and in C for the simulation program.

What we described in this paper is a six month research program where we first started off with interviews from air traffic controllers from Logan Airport in Boston,

Massachusetts. In the initial interviews we realized that we were trying to capture too much in the beginning and would ask the controller to describe the decision process with a series of planes approaching Logan Airport. However, so much information goes into a decision of delaying a plane or rerouting a plane (or simply sequencing a plane) that is far beyond the realization of the air traffic controller. He simply does it by instinct after his many years of training.

Our over-riding principle was to keep thinking of the problem as a manufacturing queing problem. Previous research has been done by one of the coauthors of this paper (Y. Pan) on manufacturing queing [12]. Instead of parts arriving at machines at certain times, we have planes arriving at an approach path at certain times.

During our research we were urged by the FAA to consider this system as a training tool for air traffic controllers. They suggested we go in this direction because the idea of an automated data link which would essentially replace the decisions of an air traffic controller did not seem feasible at this time. We agreed with them. However, the expectation is that as this work continues and as we start to add sophistication to the intelligent simulator after a period of years it will then start to be considered as an aid to air traffic controllers.

We have shown in our paper the feasibility of using an expert system for training air traffic controllers. The problem is indeed complex, however the need is significant and justifies the effort that would be entailed in going further with this research. We have shown that when two aircraft approach a decision point that we can have the computer make a decision (because it has received the expert advice of an air traffic controller) identically to the way the air traffic controller would make the decision.

The next step of the research should be to add more aircraft to the approach, and to work on the LISP C interface as well as the computer trainee interface. We believe this research builds on previous research done in the area of automated air traffic control and takes a next step forward. As air traffic continues to expand we believe the need is there for improved training through the use of qualitative simulation.

REFERENCES

[1] H. Erzberger, and L. Tobias, "A Time-Based Concept for Terminal-Area Traffic Management," presented at 42nd Symposium of the Guidance and Control Panel, NATO, Brussels, Belgium, June 1986.

[2] J.A. Kingsbury, "Basic Horizontal Resolution Equations for AERA," WP-80W00473, The MITRE Corporation, McLean, VA., June 1980.

[3] R.B. Wesson, "Problem-Solving with Simulation in the World of an Air Traffic Controller," Ph.D. Dissertation, University of Texas, Austin, Dec. 1977.

[4] S.E. Cross, "Qualitative Reasoning in an Expert System Framework," Report T-124, Coordinated Science Laboratory, University of Illinois at Urbana, May 1983.

[5] C. Shively and K.B. Schwamb, "AIRPAC: Advisor for the Intelligent Resolution of Predicted Aircraft Conflicts," Mitre Corp. Oct., 1984.

[6] O.O. Oyeleye and M.A. Kramer, "Qualitative Simulation of Chemical Process Systems," AAAI Conference, Philadelphia, PA. 1986.

[7] M. Minsky, "Frame-System Theory," in Thinking, (eds: Johnston-Laird and Wason), Cambridge University Press, Cambridge, 355-376, 1977. Also M. Minsky, "A Framework for Representing Knowledge," The Psychology of Computer Vision, New York: McGraw-Hill, 1975.

[8] Y.V. Reddy, M.S. Fox, N. Husain, and M. McRoberts, "The Knowledge-Based Simulation System," IEEE Software, pgs. 26-37, March 1986.

[9] A. Gerstenfeld, "Government Regulation Effects on the Direction of Innovation: A Focus on Performance Standards," IEEE Transactions on Engineering Management, Vol. EM-24, No. 3, August 1977.

[10] A. Gerstenfeld and Helmar Krupp, "Innovation in Systems: The Present Need," The Business Graduate, Spring 1982.

[11] A. Gerstenfeld and Paul D. Berger, "Joint Research - A Wave of the Future," Research Management, Nov.-Dec. 1984.

[12] Y. Pan, et al "A Fuzzy Production System for Multiobjective Scheduling to an One-Machine-N-Parts Problem" 23rd IEEE Conference on Control & Decision (CDC) Dec. 1984, Las Vegas, Nevada. In Proceedings of 23rd CDC.

Presenter: Sylvia P. Darnall, Dept. 067-4100

Place: Sylvia P. Darnall [2] ased Systems
General Dynamics
Workshop, Fort Worth Division or Artificial
P.O. Box 748
Intelliger Fort Worth, TX 76101 1986--Abstract

due May 1

ABSTRACT:

KNOWLEDGE ENGINEERING IN AVIONIC APPLICATIONS

One approach for obtaining expert system knowledge for a large-scale effort has been to employ automated simulated environments. A knowledge-based simulation tool can assist knowledge acquisition for expert system development in avionic applications. The performance of an expert system ultimately depends upon the quantity and quality of the domain specific knowledge available to it. Many of the situations encountered in building expert systems for avionic applications cannot be addressed by a single expert. The design of avionic knowledge-based systems for automation of mission applications presents unique opportunities to a knowledge engineer in obtaining information from multiple sources of expertise. This paper will provide a perspective of knowledge engineering approaches employing a knowledge-based simulation tool. This tool is then applied in knowledge acquisition from those multiple sources for a specific design, for example, an expert system for threat assessment. Such an avionic application illustrates the diversity of expertise that must be merged in order to build an effective expert system.

Expertise is needed from avionic systems engineers, aircrews, tacticians, and mission (scenario) planners. Each of these specialities provides a necessary but different perspective of the problem domain. Therefore, in order to interface with those specialists, a simulation environment can provide a means of overcoming some of the difficulties encountered by lack of common perspective during expert system development. In order to be most effective, the simulation must be representative of the expert's domain, as well as flexible, easily and rapidly changed, and highly interactive. A knowledge-based approach is needed in the design and implementation of this type of simulation tool.

In addition to providing a common perspective for experts from diverse backgrounds, the knowledge-based simulation tool can support a variety of dynamic and flexible scenarios to be created and tested. Some mission applications such as aircraft maneuvers can only be addressed via simulation. Others include those situations for which there is no established expertise and a knowledge engineer must draw on somewhat similar experiences and/or impressions of the expert. An example of the latter would be the development of an avionic expert system for a new future aircraft where tactics are not yet defined. Knowledge-based simulation tools facilitate not only knowledge acquisition methodology for expert system development but could also provide a means of actually developing the requisite expertise.

ABSTRACT FOR WORKSHOP ON AI AND SIMULATION

CLAIRE D. MEIER

Boeing Artificial Intelligence Center
Boeing Computer Services
P.O. Box 24346, MS 7J-64
Seattle, Washington 98124
(206) 865 - 3293

I was a member of a five person team that designed, implemented, and demonstrated in December 1985 a simulation of a free flying inspection robot that flys around the Space Shuttle directed by voice commands. The robot, known as the "Flying Eye" sends a graphic display of a three dimensional model of the Shuttle to an Apollo DN660 workstation, as seen from an orientation of a x, y, z position plus the roll, pitch, and yaw of the camera As the "Flying Eye" changes position, the view changes accordingly. The project is a first step in developing EVA (extra vehicular activity) robot, as described by NASA, to be an astronaut's assistant in the vacinity of the Space Station. The mature EVA is envisioned as a free flying vehicle equipped with manipulative arms to hold itself to a work site and perform physical tasks at that work site.

Several AI techniques were used in the implementation of this simulation The overall software architecture is that of a "Blackboard". Each of the functional components such as the voice interpreter, planner, dynamic control system, and graphics operates as a independent "knowledge source" using separate processes at a system level. The design is based on the Hearsay II architecture, although simplified in the area of triggering and data synchronization. This "Blackboard" uses multiple levels of abstraction. At the lowest level, purely numerical algorithms are being used as "knowledge sources" to calculate thrusts as a function of current commanded position. The commanded position, however, is established by a "natural language" like voice command such as "Go to Station 1". This is translated into a symbolic representation , and then into a sequence of named locations representing a path Each named location is associated with a set of coordinates which in turn generate thrusts to move the "Flying Eye" to the desired location. These steps forn a hierarchy in representing and operating on symbolic and numeric information.

The planner which is hosted on a Symbolics 3670 has the capability to "remember" how it got to its present position. Known paths can then be linked to "learn" more complicated manuevers.

Areas of future development will include: methods of obstacle avoidance using a hierarchy of models and pseudo force fields; a logical representation of the general knowledge that a space robot should know, to allow real conversation between the robot and the astronaut; and interpretation of the visual output so that the robot can tell us what it sees.

4

TOWARDS A KNOWLEDGE-BASED NETWORK SIMULATION ENVIRONMENT

Sergio Ruiz-Mier and Joseph Talavage
School of Industrial Engineering
Purdue University
West Lafayette, Indiana 47907

INTRODUCTION

The network approach to systems modeling has as an underlying philosophy to provide the modeler with simple yet powerful concepts which can then be used to capture the significant aspects of the system to be modeled. Current simulation languages such as SLAM II , Siman and INS are built around this idea and provide a set of concepts (eg. Arrival, Activity, Waiting, Routing, Departure, etc.) for model building. Yet current implementations of these languages are limited in that they do not provide explicit concepts for representing complex behavior such as decision- making encountered in many real-world situations. When complex systems need to be modeled, the user must revert to a lower level language like FORTRAN. This lack of flexibility comes from the fact that today's network languages are embedded in poor programming environments rather than this being an inherent limitation of the network approach.

This paper describes some key ideas behind SIMYON, an experimental network simulation language implemented as a subset of CAYENE, a hybrid AI programming system .

A SUITABLE PROGRAMMING ENVIRONMENT

While LISP is an excellent language for development of a basic simulation capability, it is not necessarily appropriate for incorporating extensive capabilities including representation of complex decision making. In searching for a rich programming environment in which to base a comprehensive network simulation language, it is helpful to look at different approaches to simulation that have shown some flexibility and discern which attributes are desirable. We have looked at previous implementations of frame-oriented, object-oriented and rule-oriented approaches to simulation.

CAYENE

SIMYON has been implemented as a top-level of CAYENE. CAYENE is a member of the class of programming languages known as hybrid AI systems and it is based on the idea of using object-oriented programming as a unifying principle for procedure-oriented (eg, LISP) , access-oriented (eg, demons and attached procedures) and rule based programming.

As in Smalltalk, each object (class or instance) in CAYENE is associated with a unique database containing its properties and knowledge about the object's behavior (its protocol). CAYENE's databases are different in that they are a generalization of relational databases and are regarded as logic programming environments in which properties are expressed as assertions, protocols are coded as production rules and control is through four unified programming paradigms;

— Goal directed inference based on a powerful pattern matcher.
— Object-oriented, message-passing.
— Access-oriented procedures.
— Procedure-oriented programming (LISP expressions).

At the top level CAYENE is structured as a hierarchy of objects and control is strictly by message-passing using the function

(ask <object> <message>)

where <message> is a goal to be satisfied using a backward-chaining inference procedure and the knowledge base associated with <object>.

Support for access-oriented programming is mainly through the

(if_needed <procedure>)

demon which lies dormant until there is an attempt to retrieve an object's property value. When this happens, <procedure> is executed.

One of the most common procedures used is the 'query' procedure which prompts the user for the object's property value.

Hierarchies are constructed using the inheritance functions

(is_a <superclass>)
(a_kind_of <superclass>)

and the relation function

(needs <object1> <object2> ..... <objectN>)

Finally, procedures can be constructed at the object level by using LISP expressions.

SIMYON

As we noted before, SIMYON is an experimental AI based network simulation language embedded in CAYENE.

The first step in constructing SIMYON was to generalize the message passing routine which then becomes

(ask <at_time> <object> <message>)

where <at_time> is an expression which evaluates to a number. Messages are then stored in an EVENT-FILE and sent when <at_time> matches the global variable TNOW. This generalization provides a consistent timing mechanism to drive the simulations.

The second step is to define the SIMYON system classes which are the building blocks for model construction. These building blocks are defined as objects with characteristic properties and behaviors and are arranged in a hierarchy.

To define a model using SIMYON, the user merely describes the network by initializing SIMYON system objects such as activities, branches, etc. Examples of the classic teller problem and of the scheduling of a manufacturing system are shown.

CONCLUSION

A critical need of current network simulation languages is the capability to represent complex decisions in an efficient and effective way. Simulation languages such as the SIMYON language discussed here can provide the ease of use characteristic of network languages, and at the same time incorporate user-specified decision processes in a complex and flexible format. For example, the decisions of a human expert could be represented by a rule-based expert system which would be completely compatible with the remaining network representation of the model.

The flexibility of SIMYON extends beyond its representation abilities. Simulation itself is a framework in which to perform experimentation. Yet the use of simulation in an experimentation environment calls for considerable judgment with regard to critical analysis of simulation output. Again, an expert system to control the experimentation aspects of simulation could be incorporated into the SIMYON language. Similar remarks could be made about employment of expert systems to facilitate modeling. Thus a language framework like SIMYON becomes more than just a simulation language. It really becomes a problem-solving language for a fairly broad domain of problems.

Abstract

Symbolic Explanation Systems for Computer Simulations[1]

Dr. David H. Helman [7]
201 Guilford
Case Western Reserve University
Cleveland, OH 44106

In the 1970s, the notion of a symbolic explanation facility emerged from research in artificial intelligence. Early explanation systems merely traced the behavior (in one or another way) of symbolic reasoning programs. If a user wanted to know, for example, how a program arrived at a conclusion, the program would trace the steps that led up to that conclusion. Subsequent artificial intelligence research made it clear that reasoning traces did not exhaust the kinds of information that could clarify symbolic reasoning programs. It has been suggested, for example, that a good explanation system for a symbolic reasoning program should be able to explain the strategy behind its rule orderings (Hasling, 1984). Research on explanation systems, in general, has shifted from the question "How do we explain the computational steps of the program to a user?" to "How do we explain the problem domain of the program to a user?" (note Clancey, 1983)

The problem of writing explanation facilities for computer simulations is analogous to the problem of writing such facilities for symbolic reasoning programs. An explanation facility for a computer simulation can (paralleling early ideas in artificial intelligence) trace the computations performed by a simulation in a variety of ways. It is important, however, to demarcate the other kinds of information (relating to the simulation problem domain) that can be usefully supplied by explanation facilities for computer simulations.

The first step we took in our research on explanation facilities for computer simulations was to produce a taxonomy of explanations. Our taxonomy is derived from the philosophical literature on explanations, wherein a number of types of explanations are described (see Hempel, 1965). We have been particularly interested in the literature on genetic explanation (explaining an event by citing the history of the event - note Dray, 1957), causal explanation (e.g., Salmon, 1985), what-if explanation (explaining an event by contrasting it with what might have happened - see Van Frassen, 1980), and functional explanation (explaining a variable or a component by elucidating its function within a larger system - see Hempel, 1965). We believe that these four kinds of explanations are important in a variety of explanatory activities, though in some contexts or disciplines one or another type of explanation may be paradigmatic.

The second step we took in our research was to see if the kinds of explanatory information given in genetic, causal, what-if, and how-possibly explanations could be supplied by a facility attached to a specific simulation. For this experiment, we used an inventory control model described in (Gaither, 1982). Gaither models inventory control as follows:

total annual material cost	=	annual carrying costs	+	annual ordering cost	annual acquisition cost	+	annual incoming transportation cost	+	annual carrying cost for safety stock	+	annual expected stockout cost

$$TMC = \frac{Q}{2}C + \frac{D}{Q}S + ac(D) + r(D) + (SS)C + \alpha(S')\frac{D}{Q}$$

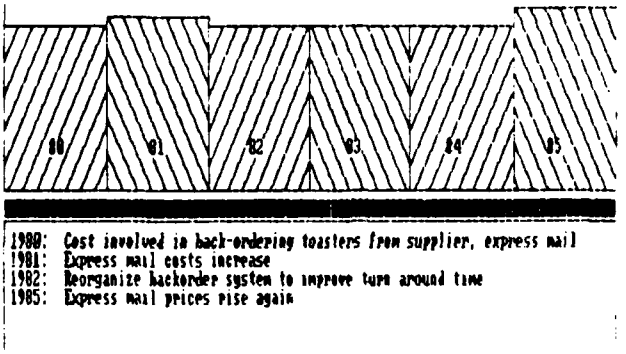
Variable Definitions

Q = fixed order quantity in units per order
C = carrying costs per unit in dollars per unit per year
D = annual demand in units per year
S = ordering or setup cost in dollars per order
ac = acquisition cost in dollars per unit
(this may be, for example, a continuous function of Q)
r = incoming transportation cost in dollars per unit
(this may be, for example, a step function of Q)
SS = level of safety stock in units
α = probability of stockout in each reorder cycle
S' = stockout, reorder costs, etc., in dollars per stockout

In a simulation based on this model, an initial estimate of the optimal order quantity (the Q that minimizes TMC) is make by calculating $1/2\left(\sqrt{2DS/C}\right)$. Q is then incremented in a range determined by this initial estimate. For each Q tested, the simulation will determine the optimal safety stock (the SS that minimizes the sum of the Annual Carrying Cost for Safety Stock plus Annual Expected Stockout Cost) by varying the estimate of α. The sample output from our implementation of the Gaither Inventory Control Model shown below only indicates the optimal Safety Stock (SS) for each Order Quantity (Q):

Order quantity (units)	Service level (1-alpha)	Safety stock (units)	Annual ordering cost	Annual carrying cost	Annual incom.ng transportation cost	Annual safety stock cost — Carrying cost	Expected stockout cost	Total cost	Annual acquisition cost	Total annual material costs
212	0.9713	19	1132.08	1066.00	26500.00	190.00	40.61	230.61	240000.00	270922.69
213	0.9713	19	1126.76	1065.00	26200.00	190.00	40.42	230.42	240000.00	270622.18
214	0.9713	19	1121.50	1070.00	26200.00	190.00	40.23	230.23	240000.00	270621.73
215	0.9713	19	1116.28	1075.00	26200.00	190.00	40.05	230.05	240000.00	270621.33
216	0.9641	18	1111.11	1080.00	27500.00	180.00	49.86	229.86	240000.00	270320.97
217	0.9641	18	1105.99	1085.00	27500.00	180.00	49.63	229.63	240000.00	270320.62
218	0.9641	18	1100.92	1090.00	27500.00	180.00	49.40	229.40	240000.00	270320.32
219	0.9641	18	1095.89	1095.00	27500.00	180.00	49.18	229.18	240000.00	270320.07
220	0.9641	18	1090.91	1100.00	27300.00	180.00	48.95	228.95	240000.00	269719.86
221	0.9641	18	1085.97	1105.00	27300.00	180.00	48.73	228.73	240000.00	269719.7
222	0.9641	18	1081.08	1110.00	27300.00	180.00	48.51	228.51	240000.00	269719.59
223	0.9641	18	1076.23	1115.00	27000.00	180.00	48.30	228.30	240000.00	269419.53
224	0.9641	18	1071.43	1120.00	27000.00	180.00	48.08	228.08	240000.00	269419.51
225	0.9641	18	1066.67	1125.00	26700.00	180.00	47.87	227.87	240000.00	269119.53
226	0.9641	18	1061.95	1130.00	26700.00	180.00	47.65	227.65	240000.00	269119.60

Genetic Explanations: For the concept of a genetic explanation (see above) to apply to the Gaither simulation, we must suppose that the simulation is run over an extended period of time, and that we have a record of the actual as well as the expected variable values. Graphing changes in variable values provides some useful information, but such graphs will not tell us what to expect in the present time period, or why significant changes occurred when they did in past time periods. In our present explanation facility, we record some of this information, because we ask the user to input text that explains variable values that diverge significantly from the values in the time period immediately past:



We are presently working on methods for constraining the symbolic explanations that may be supplied to the genetic explanation facility. This allows other modules in the explanation facility to access information from the genetic module (see Causal Explanations).

What-If Explanations: The concept of what-if explanations (see above) is implemented, in our explanation facility for the Gaither simulation, as a kind of spreadsheet. Once the user has run the simulation with a set of variable values, he may change any number of these values, and rerun the simulation. The system returns the effects of these changes on simulation results. The user may also rerun the simulation after changing the functions of the model (the user can, for example, model acquisition costs as a step function of order quantity). We are in the process of altering this module to present to users qualitative what-if explanations.

Causal Explanations: Our causal explanation facility has two parts, a functional trace facility and a symbolic reasoning mechanism. The functional trace facility answers the question "What caused the simulation to arrive at value V for quantity Q?" This question is answered by (qualitative) traces of the computations that led to the calculation of V for Q. The symbolic reasoning mechanism answers the question "Why did the simulation result (TMC) change in the way it did from the previous year to this year?" Our first step in anwering this question is to calculate numerically the effect of individual changes in variable values on total change in TMC (the technique is similar to partial differentiation - see Kosy, 1984). We can, for example, after this calculation, conclude that change in acquisition costs, on their own, would have accounted for half of the total change in TMC. Why,

however, did acquisition costs change as they did?

In our causal explanation module, we use a backtracking rule-based reasoning program to determine the causes of changes in variable values (e.g., some top-level goals in the program are the reasons acquisition costs might have changed as they did). What is perhaps more interesting about this module is that we have interleaved the symbolic reasoning program with information derived from the Gaither simulation. In most symbolic reasoning programs, conclusions are reached by 1) deduction, or 2) asking the user a question. We can, however, use the simulation and the simulation explanation facility as a kind of "oracle" for a symbolic reasoning mechanism. Rising inflation, for example, might account for rising acquisition costs, if these costs have, in the last several years, tracked inflation.

Functional Explanations: In the context of simulation explanation facilities, the goal of functional explanation is to clarify simulations by elucidating the functions of simulation model components. We distinguish between derived concepts, which are the results of simulation computations, and basic concepts (i.e., constants or values input by the user). In our functional explanation system for the Gaither simulation, which is in the process of being implemented, users may ask for definitions of any of the basic or derived concept used in the model. Basic concept definitions (e.g., the definition of demand) give an intuitive characterization of the concept and describe (where it is appropriate) how one might estimate its value. In our definitions of derived concepts, we describe how the derived concept is computed, and we explicate the presuppositions (and limitations) of its characterization in the simulation model. Our definition of the derived concept Annual Carrying Costs, for example, notes that the model assumes demand is linear and that order quantity does not change from cycle to cycle. We are also working on integrating model comparisons into our functional explanation system, so that the user can understand where the presuppositions of other inventory control models differ from the presuppositions of Gaither's model.

Future Directions: We have recently started two projects which extend the research described in this abstract. First, we are transferring the rule-based portion of our present explanation system to a PROLOG-based explanation shell (see Sterling, 1985). Second, we would like to test our theory of explanations empirically. We have, in conjunction with Professor Elizabeth Short of the CWRU Department of Psychology, designed and are in the process of implementing an experiment to explore the relationship between increased understanding of the Gaither model and the use of one or another module of our explanation facility.

Notes

[1] This research was supported in part by the core research program of the Center for Automation and Intelligent Systems Research through use of computing facilities.

[2] Dr. David H. Helman is an Assistant Professor in the Department of Philosophy and a Staff Member in the Center for Automation and Intelligent Systems Research.

Bibliography

Clancey, W. (1983). The epistemology of a rule-based expert system--a framework for explanation. Artificial Intelligence, 20, 215-251.

Dray, W. (1957). Laws and Explanations in History. Oxford: Oxford University Press.

Gaither, N. (September, 1982). Using computer simulation to develop optimal inventory policies. Simulation Magazine.

Hasling, D., Clancey, W. & Rennels, G. (1984). Strategic explanations for a diagnostic consultation system. Int. J. Man-Machine Studies, 20, 3-19.

Hempel, C. (1965). Aspects of Scientific Explanation. New York: MacMillan.

Kosy, D.W. and Wise, B.D. (1984). Self-explanatory financial planning models. Proceedings of the National Conference on Artificial Intelligence.

Salmon, W. (1984). Scientific Explanation and the Causal Structure of the World.

Sterling, L. (1985). Explaining explanations clearly. Case Western Reserve Center for Automation and Intelligent Systems Research Technical Report 124-85.

van Frassen, B. (1980). The Scientific Image. Oxford: Oxford University Press.

MODVEX

An Expert System for model validation, maintenance,

coordination and development[1]

James M. McKinion [8]
U. S. Department of Agriculture
Agricultural Research Service
Crop Science Research Laboratory
P.O. Box 5367
Mississippi State, MS 39762-5367

MODVEX provides an expert system shell for the simulation models of the Crop Simulation Research Unit (Acock,1985; Baker,1983; Baker,1985). MODVEX serves as an inner shell for applying Artificial Intelligence and Expert System Technology to the development, testing, and operation of crop and insect models. The outer Expert System shell is the COMAX system (Lemmon, 1986; McKinion, 1985). COMAX is a crop management expert system which exercises crop models to find the optimum crop management strategy.

The MODVEX Expert System shell provides the following capabilities: (1) MODVEX provides for automatic validation of the model; (2) MODVEX serves as a system for maintaining the proper operation of the models; (3) MODVEX provides for automatic coordination of process-level modules when a model is called for execution; and (4) MODVEX serves as a model development tool when modules are called for and do not exist in the module code base.

1/ A contribution of the USDA-ARS Crop Simulation Research Unit in cooperation with the Agronomy Department, Mississippi State University and the Mississippi Agricultural and Forestry Experiment Station.
2/ The author is: James M. McKinion, Electronics Engineer, USDA-ARS Crop Simulation Research Unit, Mississippi State, MS.
The use of company names and brand names is for information only and does not represent an endorsement or warranty by the USDA.

MODVEX is initially being used for crop model development. When insect simulation models becomes available, MODVEX will be extended to handle interactions of the crop models with the insect models.

Validation

Crop models can be checked for validity by applying the MODVEX system. Standard validation data sets are used in the system's knowledge base. The rule base incorporates the heuristic knowledge of the Crop Simulation Research Unit modelers. The user of MODVEX can then run MODVEX to automatically test the validity of the crop model each time improvements are made in the model code to insure integrity. MODVEX generates a report which delineates whether or not the model passes the validation test. If the model fails to pass the test, MODVEX uses its knowledge of model structure and operation to determine the source of failure and generate in its report the likely source. Once the model can be tested for validity by the computer automatically, many opportunities arise for the system user.

Development

One of the first uses of this validation checking capability is to use MODVEX as a model development system. Crop models have a number of parameters which are either very difficult or impossible to measure. However, the numerical range in which the parameter must lie is known from thermodynamics, physical properties, or physiological properties. MODVEX can be used to automatically find the set of parameters which optimize the predictive performance of the crop model. The rule base of heuristic knowledge of model development is used to dynamically prune the search

space so that exhaustive search is not used and the model's parameters can be determined quickly and economically in terms of computer execution time. Knowledge of model structure, process structure, soil physics, plant physiology, and micrometeorology were used to build the knowledge base which provides this capability.

Maintenance

Maintenance in the sense used here means proper operation of the model. MODVEX serves as a data filter to ensure the proper operation of the crop model. As such, MODVEX sits between each module in the operational system and monitor the values passed from module to module. Again, knowledge of the the model-plant-soil system is used. When critical state variable values begin to deviate from their expected ranges, the system user is given a warning that the simulated result may not be valid. If state variables deviate strongly from their expected ranges, the system can be shunted into a "Debug" mode and the user offered a list of actions to take: (1) Terminate run, (2) User provide value, (3) MODVEX to provide value, or (4) Continue run as is. The maintenance mode here serves much like a truth maintenance capability.

Coordination

As crop models are reduced to module form in relation to plant and soil processes and as different plant species are modeled, MODVEX serves as an Expert System shell to properly choose the appropriate modules to link together to generate the chosen plant model. This becomes increasingly

important as more and more species dependent processes are described in module form. In fact it performs almost like a database query system which will pick the appropriate modules and submodules in response to selection by menu or lists of attributes, except in the case of MODVEX the knowledge base is required for choosing the proper modules, if they exist, and informing the user if they do not and whether or not substitutions can be made and with what degree of error can be expected if substitutes are made.

Application of MODVEX

MODVEX is currently under development for the cotton model GOSSYM (Baker, 1983). Extensions of MODVEX are planned for the GLYCIM soybean crop model (Acock, 1985) and the WINTER WHEAT crop model (Baker, 1985). The crop models have been developed on a DEC VAX 11/750 superminicomputer using ANSI 1977 FORTRAN. MODVEX has been developed on a Symbolics 3670 LISP Machine using Inference Corp.'s Automated Reasoning Tool (or ART). In operation on the Symbolics 3670, MODVEX uses the LISPFUNCTION extension to FORTRAN to call LISP functions which actually activate ART from the LISP world. Thus the FORTRAN program calls LISP functions which initiallize ART, load the knowledge base from disc, resets the knowledge base, asserts data into the knowledge base from the running FORTRAN program, and runs ART which then performs the above functions. The MODVEX tool could be the beginning of a revolution in crop modelling.

References

Acock, B., Reddy, V.R., Whisler, F.D., Baker, D.N., McKinion, J.M., Hodges, H.F., and Boote, K.J., 1985. The soybean crop simulator GLYCIM model documentation 1982 (PB 85171163-AS). U.S. Department of Agriculture, Washington, D.C. Available from NTIS, Springfield,

Baker, D.N., Lambert, J.R., and McKinion, J.M., 1983. GOSSYM: a simulator of cotton growth and yield. S.C. Agric. Exp. Stn. Tech. Bull. 1089, 134 pp.

Baker, D.N., Whisler, F.D., Parton, W.J., Klepper, E.L., Cole, C.V., Willis, W.O., Smika, D.E., Black, A.L., and Buaer, A., 1985. The development of winter wheat: A physiological process model. In ARS Wheat Yield Project. June,1985.

Lemmon, H.E., 1986. Cotton crop management expert system - COMAX. Science. In press.

McKinion, J.M. and Lemmon, H.E., 1985. Expert systems for agriculture. Computers and Electronics in Agriculture, 1(1985)31-40.

Mckinion, J.M. and Lemmon, H.E., 1985. Symbolic computers and AI tools for a cotton expert system. ASAE Paper #85-5520, 16 pp.

9

ARMY-NASA AIRCREW AIRCRAFT INTEGRATION PROGRAM

Executive Summary
A^3I Program Plan

Introduction

The purpose of the A^3I program is to provide a human factors predictive methodology for use by designers of cockpits and training systems for future advanced technology rotorcraft which will result in the production of cost- and performance-effective man-machine systems. A specific outcome will be to reduce the risk and unexpected cost of errors early in the conceptual and developmental process, and to aid in the evaluation and trade-off analyses of proposed systems.

The A^3I program is an Army-NASA exploratory development program with the purpose of developing a rational predictive methodology for helicopter cockpit system design, including mission requirements and training system implications, that integrates human factors engineering with other vehicle/system design disciplines at an early stage in the development process. The program will produce a prototype Human Factors/Computer Aided Engineering (HF/CAE) workstation suite for use by design professionals. This interactive environment will include computational and expert systems for the analysis and estimation of the impact of cockpit design and mission specification on system performance by considering the performance consequences from the human component of the system. The technical approach is motivated by the high cost of previously unavoidable redesign and retrofit to suboptimal systems, the ever-increasing cost of training systems, including simulators, and the loss of mission effectiveness and possibly lives due to ill-conceived man-machine design. The methodology developed to achieve the goals of the A^3I program might be generalized as a paradigm for the development and planning of a variety of complex human operated systems.

Stephen Lakowske
NASA/Ames Research Center
MS 239-19
Moffett Field, CA 94035
415-694-6436

10.

Some Cognitive Constraints On Simulation Aids For Programmers

David C. Littman & Elliot Soloway

Cognition and Programming Project

Department of Computer Science

Yale University

New Haven, CT 06520

The research reported in this paper was cosponsored by the Personnel and Training Division Research Groups, Psychological Sciences Division, Office of Naval Research and the Army Research Institute for the Behavioral and Social Sciences, under Contract No. N00014-82-k0714, Contract Authority Identification Number 154-492.

For the past three years we have been studying expert software designers and maintainers with the goal of understanding some of the major cognitive processes that support expertise in computer programming. One of the most intriguing findings of our work is the extent to which both designers and maintainers rely on *mental simulation* to develop their understanding of computer programs.

- When program designers are in the process of developing a program to perform some tasks, they mentally simulate their programs to check the behavior of their evolving designs against a description of what the program is intended to do.
- When program maintainers have the task of making an enhancement to a program that they have never before worked with, they spend a great deal of time mentally simulating the program to figure out how the program is organized and what it does.

Since mental simulation appears to be so important for both program design and program maintenance, it seems reasonable that program development software should *assist designers with mental simulation*. But what *is* mental simulation of programs? How does it help programmers? And what should program development software do? We want to suggest some tentative answers to these questions.

Our approach to understanding mental simulation has been empirical: we have studied more than 30 expert, professional software designers and maintainers as well as many novices. Our experimental methodology presents programmers with design or maintenance tasks that take approximately three hours to perform. We videotape the programmers as they go about their tasks while an interviewer prompts them to maintain a running commentary on their problem solving. We then analyze the transcripts of the sessions for evidence of various kinds of cognitive activity, such as mental simulation of programs.

As we analyzed the protocols of the novice and expert programmers, we identified five interesting points about programmers' mental simulation of programs:

- First, programmers mentally simulate programs in order to build *mental, causal models of programs*.
- Second, there are *at least two kinds* of mental simulation.

- Third, programmers *coordinate the knowledge* provided by both kinds of simulation in their causal models of programs.
- Fourth, simulation is very useful for reasoning about *incomplete programs*.
- Finally, and perhaps most relevant to the issue of constructing useful tools for programmers, novices did not perform mental simulation as effectively as experts.

Building Mental Models of Programs

Designers and maintainers mentally simulate programs in order to build mental models of them. That is, they use simulation to acquire knowledge that permits them to construct mental models that relate the *function* of the program to its *structure*. By performing mental simulation, programmers discover the "mapping" between *what* the program does (its function) and *how* it does it (its structure.) While both designers and maintainers use simulation to build their mental models, they seem to use it in "opposite directions." Designers go primarily from the *intended function* of a program to the construction of pieces of structure to add to the their mental models (and then to the program itself); maintainers appear to add pieces of functionality to their mental models as they discover how the existing structure of the program embodies its functions.

Two Kinds of Mental Simulation

Our analysis of the behavior of designers and maintainers revealed two levels of mental simulation. Rather than using a single monolithic simulation skill, programmers use different kinds of simulation to acquire different kinds of knowledge. First, *structural simulation* is based on tracing data flow and control flow. Structural simulation provides the programmer with a detailed mental model of how the components of the program behave when it runs; structural simulation does not say *why* the program does what it does, only *what* it does. In contrast to structural simulation, *functional simulation* gives the programmer a view of the program in terms of the *goals it achieves*. Rather than generating information about *what* the program code does, functional simulation gives the programmer answers to questions about *why* the program is

constructed as it is. Instead of being tied to the low level of data flow and control flow, functional simulation is based on tracing what we call *goal flow*. Tracing goal flow gives the programmer information about how the program achieves its functional goals (i.e., what *plans* it uses), and how subgoals cooperate to achieve larger goals.

Coordinating The Two Levels of Simulation

Of particular interest is the observation that programmers appear to use both structural and functional simulation *together* to build their mental models of programs. Our current hypothesis about why they use both kinds of simulation is that the different kinds of knowledge the two levels produce are necessary to understand *interactions* among components of the program. A mental model of just the data flow and control flow level, provided by structural simulation, does not *by itself* permit the programmer to understand all interactions since many arise because program *goals* must be coordinated (e.g., several subgoals must be coordinated to achieve a major goal.) Conversely, if a programmer's mental model of the program contains *only* functional knowledge, then crucial aspects of the program's runtime behavior that depend on control flow and data flow interactions (e.g., satisfaction of preconditions,) may not be understood. Thus, in order to understand interactions, programmers appear to need a mental model that coordinates information from both the structural and functional levels. In fact, we have shown that programmers who have mental models of programs that coordinate structural and functional levels are more successful with program enhancement tasks than programmers who may have both kinds of knowledge but who fail to coordinate them in their mental models of the program.

Mentally Simulating Incomplete Programs

A programmer does not need a complete program in order to simulate it mentally. For example, we observed expert program designers engage in functional simulation as soon as they had an initial, sketchy idea of how their programs should accomplish the main goals they wanted their programs to achieve. As they filled in their designs, their functional simulations became

more detailed and provided them with more precise information for evaluating whether their designs were meeting the requirements. In addition, as they defined the manner in which modules communicated through parameters, and decided on the execution order of the modules, they began to perform structural simulations to evaluate control flow and data flow.

Simulation Is Part of Expertise

Not all our programmers simulated their programs. In fact, novices either failed to simulate them or did not do so with the same skill as experts. Novices' poor simulation appeared to be attributable to three deficits:

- Novices did not know *when* to simulate programs. While they performed some mental simulation, they appeared unable to determine when it would provide them with information necessary to build a good mental model of the program.
- Novices did not know *what parts* of the program to simulate to obtain knowledge they lacked. Sometimes novices wanted to discover how a part of the program worked and performed simulation to do so. Frequently, however, they could not identify the relevant parts of the program to simulate to answer the questions that led them to attempt simulation.
- Novices did not know *how* to simulate programs to build good mental models. When novices identified relevant parts of the program to simulate, they failed to simulate them with, for example, informative test cases. In addition, they often failed to make correct inferences about the state of the program even when they did perform the correct simulation.

In contrast, experts almost always knew when to simulate, what parts of the program to simulate, and they were very astute at drawing appropriate inferences from their simulations.

Conclusions

What does all this mean for *cognitively based support systems* for programmers? We believe that our empirical approach to studying the cognitive activities of programmers yields useful information that can provide constraints on the design of intelligent support systems for programmers. At the very least, our data suggest that programmer support systems will have to take account of the central role simulation plays in program development and enhancement. It seems to us that a major issue implied by the importance of mental simulation is *how to help programmers use simulation to build their mental models of programs*. Based on our

observations of experienced programmers, we believe that support systems will have to 1) help programmers perform both structural and functional simulation and 2) help them coordinate the results of simulations.

In order to help programmers coordinate simulations at the structural and functional levels, support systems will need to address the critical problem of *how* to give the programmer the information that he or she needs *when* it is required. Instead of simply inundating the programmer with information, the support program must provide the "right information" at the "right time." We currently believe that the most reasonable approach to this problem is itself a knowledge-based simulation tack. A cognitively based programmer support system will have to simulate the user's reasoning so as to understand the user's emerging mental model of the program, determine what information would help the user to develop a "correct" mental model of the program, and then provide it.

Modeling of a Military C^3 (Command, Control, and Communication) System Using Discrete Event and AI Techniques

D. J. Roberts
J. D. Morgeson
J. S. Dreicer
H. W. Egdorf

Los Alamos National Laboratory
P.O. Box 1663
Los Alamos, New Mexico 87544

ABSTRACT

With their inherent complexity, military C^3 systems are difficult to model. Because of the enormous quantities of communications traffic and transient data generated by a military operation, the human decision-making process is not always conducted with full utilization of all available data. Because of the transitory nature of data in these operations, it can be difficult to define the rules that the human decision-makers use.

Operating under funds supplied by the Los Alamos National Laboratory Institutional Supporting Research and Development (ISRD) 1986 Program, the authors have developed a simulation of a proposed military C^3 system. The purpose of the project was two-fold. First, the authors wished to learn the potential of the selected simulation environment for simulating C^3 systems. Second, if successful, the prototype simulation would be used to demonstrate the design methodology for producing such simulation systems.

I. The Simulation Environment

The desired simulation environment would blend traditional discrete event simulation techniques with expert system-like capabilities to execute the complex decision-making events. In order to realistically represent the behavior of a military operation, the model must have the ability to maintain the "ground truth" regarding the operation, as well as the individual entities' perceptions of the truth. Further, the entities must be able to learn, as facts regarding the operation become known to them. For example, a town containing a fuel depot might have been destroyed by the enemy. However, this fact is unknown to an entity in the simulation requiring fuel. His decision process regarding where to send a re-fuel team could therefore cause him to dispatch his team to the destroyed depot. Upon arrival, the re-fuel team would then have access to the "ground truth" concerning the status of the depot, and his perception of the depot's status can be updated.

The hardware that was selected for this project was a Symbolics[1] 3600 with 1 megabyte of

[1] Symbolics is a trademark of Symbolics, Inc.

memory and a 380 megabyte disk. KEE[2] was chosen as the software with which to implement the simulation. Utilizing KEE's ability to interface with the Symbolic's native Zetalisp, a discrete event simulation controller was written in LISP to drive the simulation. The complex C^2 (Command and Control) events were written in KEE's RuleSystem2. The less complex physical events (such as a refueling operation) were coded in LISP and invoked as KEE methods. The suite of actors (entities) within the model were defined by KEE units.

2. The Physical System That Was Modeled

One of the military systems that has been proposed for national defense is the Midgetman Mobile Intercontinental Ballistic Missile (MICBM) system. The system as proposed would consist of a number of mobile launcher organizations that could be deployed throughout a region of the United States. A launcher organization would be comprised of the launcher vehicle and other support equipment and personnel.

Since the MICBM system does not exist, the authors "designed" the hypothetical MICBM system, specifying the physical assets and attributes of a launcher organization. The following is a list of the physical equipment comprising a launcher organization, and the assets and attributes of the equipment:

Vehicles (5 types)	
1	Lift capacity, tons
2	Fuel capacity, gallons
3	Fuel type
4	Fuel consumption, MPG
5	Speed, MPH
6	Mean Time Between Failures (MTBF)
7	Vehicle hauling capacity (cubic feet and tons)
8	Vulnerability number (used to measure the vehicle's susceptibility to explosion blast effects)
9	Radiation protection factor

Electrical Generators (4 types)	
1	Capacity, kW
2	Fuel consumption, gal/hr
3	Fuel type
4	Weight, tons
5	MTBF, hours

Radios (5 types)	
1	Minimum and maximum broadcast range
2	Minimum and maximum frequency range
3	Power consumption, kW
4	Type of data, digitized (BAUD), voice

[2] KEE (Knowledge Engineering Environment) is a trademark of IntelliCorp of Mountain View California.

ADP Equipment (2 types)	
1	Electromagnetic Pulse (EMP) vulnerability
2	MTBF
3	Power consumption, kW

3. Actors (Entities) in the Model

The following is a list of the suite of actors, or entities that the model uses to represent the proposed MICBM system:

(1) **Launcher Organizations**, 1 to N may be represented within the model.

(2) A **Command Actor** represents a fixed hardened command post. Other actors (**Launcher Organizations, Supply Teams, Reconnaissance Teams**) communicate with the **Command Actor** via radio or land line. This actor can grant or deny requests from other actors, or order other actors to action.

(3) A **Threat Actor** represents enemy intelligence. It possesses a suite of sensors (overhead detection systems, radio direction finding systems, etc.) that can detect **Launcher Organizations** or other mobile actors, or fixed depot sites. Upon detection, this actor has the capability to launch a nuclear attack against the actor, or a depot.

(4) A **Terrain Advisor/Master** performs two functions. First, it maintains the ground truth regarding the "world" of the model. All information regarding sites, routes, actors, etc., damaged or destroyed by conventional or nuclear attack is maintained by the **Terrain Master**.

Second, the **Terrain Advisor** responds to an actor's request for route information utilizing the actor's perceptions of the "world".

(5) A **Pseudo-Actor** represents an information net whereby observations made by civilians, ham radio operators, police, etc., can be broadcast to any actor within receiving range. These observations may be of damaged or destroyed sites, cities, highways, or other actors.

4. Operational Concept and Problem Statement

To the largest extent possible, it was desired that the design of the simulation allow the specification of the operational concept to remain flexible and easily changed. This was accomplished by defining the C^2 (decision-making) events in KEE's RuleSystem2. The natural-language syntax of KEE's RuleSystem2 is more easily learned and understood than LISP code. In this way the client can change any aspect of the operational concept by simply changing the rules defining the event of interest.

The problem statement to be answered by this prototype simulation is as follows:

How survivable and effective is the MICBM system in the event of global nuclear war?

To answer this question, the model reports the following information:

(1) Numbers of launchers damaged or destroyed before they could carry out their mission,

(2) Numbers and types of personnel losses,

(3) Amount of communications traffic that was not received, and

(4) Information regarding enemy detection.

5. Summary

The Symbolics/KEE environment is richer than other traditional simulation environments such as SLAM, SIMAN, GPSS, and SIMSCRIPT. Combining an artificial intelligence shell like KEE with a powerful object-oriented programming language like Zetalisp can result in a discrete event modeling capability superior in many ways to others currently in existence. The authors are satisfied with the results of the prototype MICBM simulation and are preparing general "how-to" documentation for designing such simulations.

However, there are some disadvantages to this environment. First is the speed (or lack thereof) in model execution. Second is the lack of any of the discrete event modeling tools such as a simulation controller or a library of probability density functions.

However, for those systems developers who don't mind having to write their own controller, gather their own queue statistics, and aren't primarily interested in model execution speed, this environment provides a powerful modeling capability, well suited to the C^3 class of systems simulation.

Colleen Oresky [13]
ESL
495 Java Drive
P.O. Box 3510
Sunnyvale, CA 94088-3510

STRADS OVERVIEW

In April 1985, the first demonstration of the Strategic Automatic Discovery System (STRADS) was achieved. This first phase system consisted of an application of Douglas Lenat's machine-learning technique, EURISKO, to the problem of generating Middle East strategic scenarios as a potential tool for geopolitical analysts. EURISKO had already been applied with promising results to several other domains, including:

- Championship designs of naval fleets (winning a national tournament against human competitors the two years in which EURISKO was allowed to enter the tournament)
- Discovery of novel, viable 3D VLSI designs
- Discovery of mathematical concepts

The ultimate contribution of Discovery systems in the present application -- generation of strategic scenarios -- is that of conceptual blockbusting: aiding human analysts in the difficult, time-consuming and crucial process of developing a range of useful scenarios to support policy and thereby help to reduce surprise.

The application itself breaks new ground in the use of simulation technology to aid geopolitical analysts. As has been well documented, the human analyst is beset by a number of cognitive, epistemological and institutional constraints on scenario generation: the symptoms are frequently expressed as "mind set," "bias," "mirror imaging"

and the like. Further, the global geopolitical problems confronting these analysts tend to defy mathematico-statistical approaches -- for example, the historical failure of various probability based attempts to predict international events at analytically meaningful levels of detail and which also are credible and actionable for analysts prior to the event. More to the point is a system such as STRADS which raises the readiness of analysts by powerfully searching for, and discovering, characteristic scenarios for various times and places, this allowing a certain cognitive rehearsal and exercising which raises analysts' ability to interpret new signals of actual events during time-critical periods. Hence the measure of effectiveness for STRADS is less directly that of providing predictive accuracy than that of promoting analytic readiness. This is not to overlook the fact that a discovery system of great power can help considerably to safely narrow and focus the vision of the human geopolitical analyst to realms of highly likely eventualities. Nor is it to obscure the basic ability of a machine learning system efficiently to discover new heuristics for geopolitical analysis.

Overview of STRADS 1. In sum, the system developed under Phase 1 entailed these efforts:

Knowledge Acquisition. As a first step, project team members performed research into political, military, economic, cultural, meteorological and other characteristics of a selected set of Middle Eastern and other (such as U. S. and USSR) actors. In preparing the knowledge base, project team members consulted geopolitical analysts working for the Commander-in-Chief, Pacific, and responsible for monitoring developments in the Middle East. Several additional Middle East specialists also reviewed the knowledge base.

Knowledge Base Structuring. The second step was to enter into the computer a working knowledge base sufficient to allow demonstration of concept feasibility. This machine knowledge base consisted of a frames-and-slot representation in which chief military, political, economic and cultural characteristics of over a dozen countries and organizations associated with the Middle East, together with the U. S., U.S.S.R., Japan, NATO and other countries and organizations, were represented. There were over 450 individual slots describing each actor.

Development of Rules. Families of Rules -- at present there are about one hundred rules -- to permit scenario generation were then developed. These consisted of several rule-types, including:

1. Natural Event Rules: governing environmental events (earthquakes, etc.), deaths of leaders, and other events.

2. Stopping Rules: governing the termination of scenarios as a function of violation of time assessments and/or occurrence of pre-defined events (e.g., the onset of certain types of conflict).

3. Reacting Rules: governing causal interactions among actors in developing scenarios (e.g., military readiness reactions of one country to hostile activities in another).

4. Rippling Rules: governing the proliferation of effects of local events out to other actors in the knowledge base (e.g., effects of local economic changes on region at large).

Development of Control Structure. The relatively complex control structure cannot be explained in the present allocated space. However, I note that this complex control structure is designed to meet future requirements as the rule sets and knowledge base increase in size and sophistication. It is best discussed here with reference to these major aspects.

1. Time: The simulation time interval varies depending on the characteristics of the scenario. A large time gap may occur when no events are scheduled. However, smaller increments are required when there is a lot of activity.

2. Rule Organization: The rules have been organized hierarchically by type. This organization allows the system to control which rules are considered at particular points during the scenario.

3. Displayed Information: During scenario generation, we have reasonable control over what information is displayed to the user. This will allow us to build customized user interfaces during Phase II.

Development of a Demonstration Scenario-Generation Capability. The final task in developing the prototype system was to generate initial scenarios to explore concept feasibility and application.

AI AND SIMULATION AT TASC

Marc R. Halley [14]
The Analytic Sciences Corporation
8301 Greensboro Drive
Ste. 12000
McClean, VA 22102

For the past three years, the AI group at TASC has been involved in applying AI and simulation techniques to the problems of collection and resource management. Collection management involves allocating and tasking multiple sensors to collect selected information. This task requires knowing tremendous amounts about the information to be collected plus knowing how to effectively allocate resources to collect the most important information.

We have developed a series of resource advisor programs which have dealt with mission planning, situation assessment, and real time mission execution. I will describe the mission planning system which combined AI expert system techniques with mathematical programming and simulation.

The TASC Mission Planner Expert System is a "multi-level" system which gives advice about mission planning. It combines a "surface level" model of production rules with a "deep" model of mathematical optimization and simulation.

Every day the mission planner must develop a deployment of resources to collect the information he is tasked to collect. The mission planner faces a variety of issues in choosing a deployment: tasking priorities, resource constraints, geography, time, collection objectives, and collection characteristics. Previous attempts at decision aids for collection management used mathematical programming to develop complicated "deep" formal models to allocate resources to tasks.

However, several problems occurred. The mathematical models did not account for all the input variables and are so complex that it took a mission planner 5 hours to set up a simulation run.

An expert system was built to aid the planner in developing resource deployments. (Figure 1) Expertise was taken from mission planners and encoded as production rules. These production rules take as input an intended collection goal, and on the basis of expert knowledge, produce a deployment plan. The deployment plan is based on heuristics and therefore, may be only 80% acceptable and not mathematically optimal. The computer generated deployment plan is then passed into the "deep" mathematical simulation to produce a more precise plan. In most cases, however, the surface level production system proved to be just as good as human experts, and reduced the time to develop a deployment plan from 5 hours to 10 minutes.

The production rule system was built on a Xerox 1108 LISP machine using Interlisp-D and the KEE knowledge engineering tool.

The mathematical model was written in FORTRAN on a VAX 11/780. The Xerox 1108 was connected to the VAX via an Ethernet local area network. Plans generated on the Xerox were shipped electronically to the VAX where detailed mathematical treatment was performed. The total plan generation scenario was reduced from 5 hours to 10 minutes.

The expert system had 6 parts. (Figure 2) The planning sequence started with Menu oriented entry of collection goals. These collection goals were then passed to a planning and inference module which contained 190 rules and 270 frames and produced a deployment plan. This plan contained the allocation of resources to collection goals.

The reasons for each of the results could be explained using a rule tracing capability. The plan could also be modified by the human mission planner if he thought that selected portions of the computer generated plan were not correct.

The human edited deployment plan can then be sent across the Ethernet to the VAX and entered into the simulation program. This program simulated the proposed resource deployment and calculated the consequences of the proposed decisions. The human mission planner can then continually modify portions of the simulation for fine tuning. The AI program also contained a module to acquire new information from the mission planner. If any collection goals were not known, the system would prompt and build new frames to put in the knowledge structure.

The FORTRAN simulation used the techniques of operations research, notably mathematical optimization and allocation.

The mission planning expert system (Figure 3) used multiple AI programming techniques: frame based knowedge representation with multiple inheritance, rule based reasoning and explanation, object oriented programming, and bit map graphics. Approximately 6 man months were spent acquiring the knowledge to do planning.

The expert system was later evaluated against human planners to determine if the computer generated plans reproduced expert knowledge. (Figure 4). Test cases were given to 2 human experts and the computer; answers were generated and given back to the same human experts for evaluation. The answers were evaluated on a scale of 1 to 6. A score of 1 meant unacceptable deployment, while a score of 6 meant a perfect deployment. A score above 3 meant an acceptable answer, while a score below 3 meant an unacceptable answer.

In 16% of the cases the humans gave the computer a higher score than the humans. In 50% the computer and humans tied, and in 32% of the cases the computer was scored lower than humans. In no cases, however, was the computer score rated unacceptable (less than 3), while in one case the humans rated their own solution unacceptable (less than 3). It also must be remembered that the human took an hour to generate a plan while the computer took 3 minutes.

Figure 1.

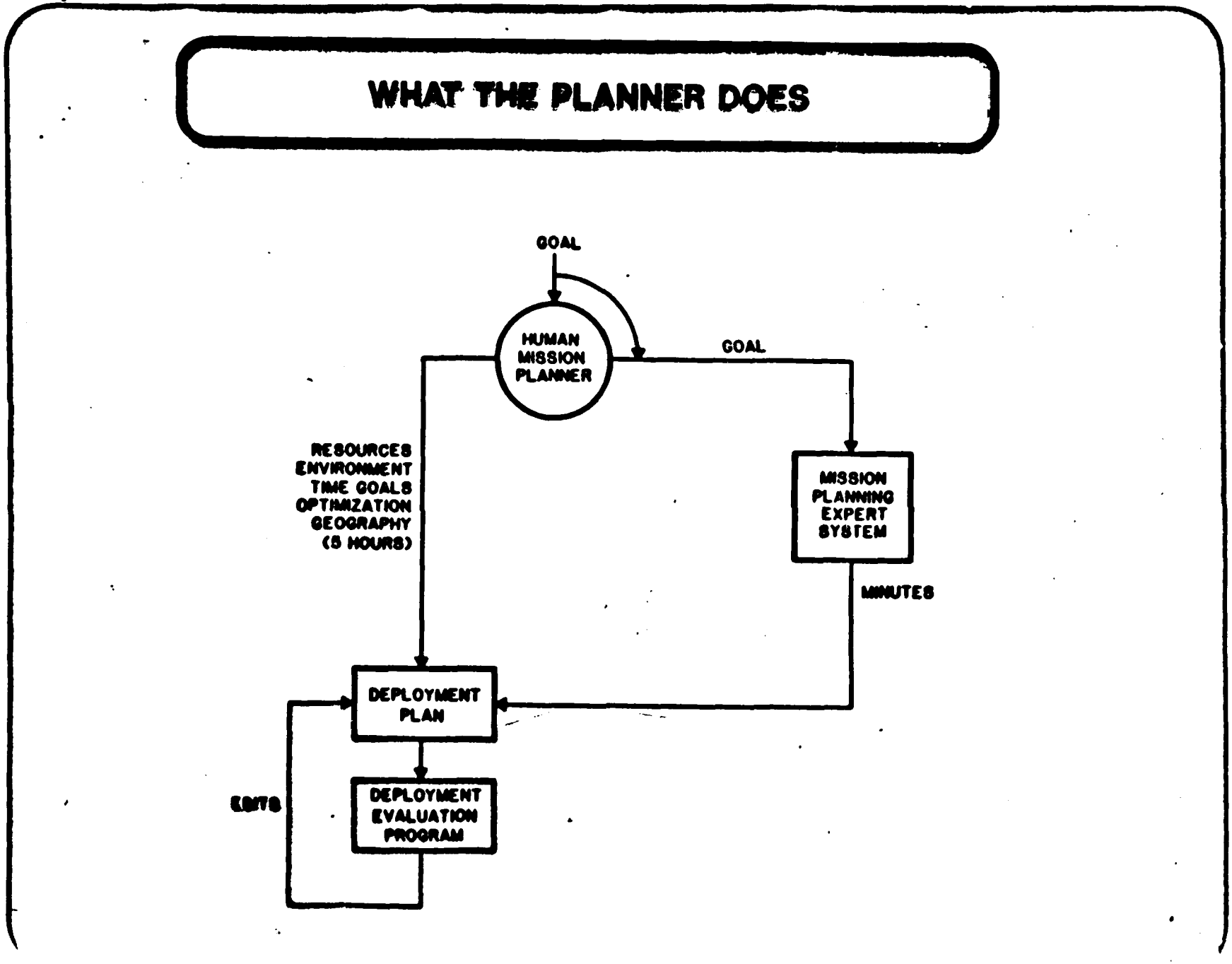

WHAT THE PLANNER DOES
GOAL
HUMAN MISSION PLANNER
GOAL
RESOURCES
ENVIRONMENT
TIME GOALS
OPTIMIZATION
GEOGRAPHY
(5 HOURS)
MISSION PLANNING EXPERT SYSTEM
MINUTES
DEPLOYMENT PLAN
EDITS
DEPLOYMENT EVALUATION PROGRAM

Figure 2.

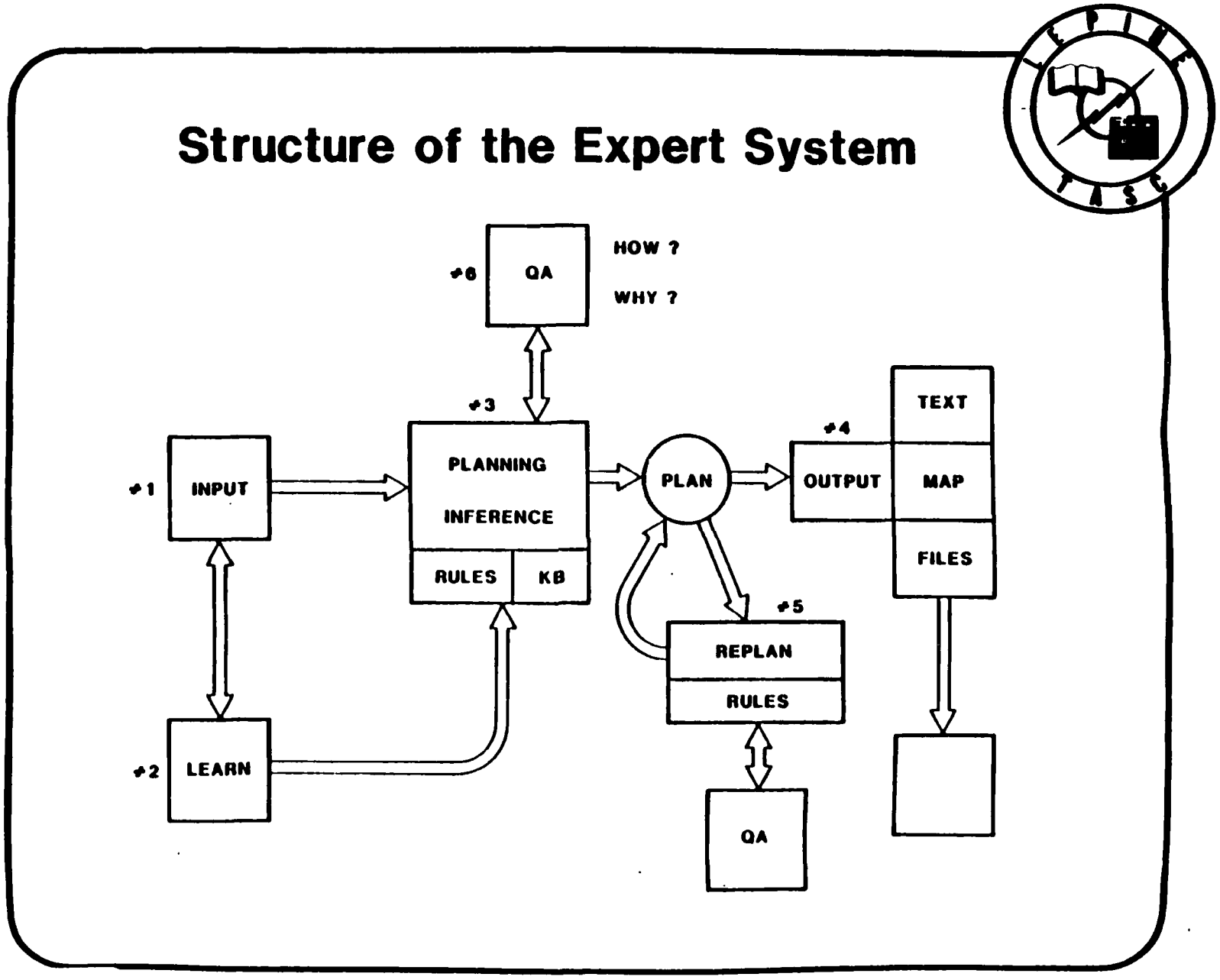

Structure of the Expert System
#6
QA
HOW ?
WHY ?
#3
#1
INPUT
PLANNING
INFERENCE
RULES
KB
PLAN
#4
OUTPUT
TEXT
MAP
FILES
#5
REPLAN
RULES
#2
LEARN
QA

Figure 3.

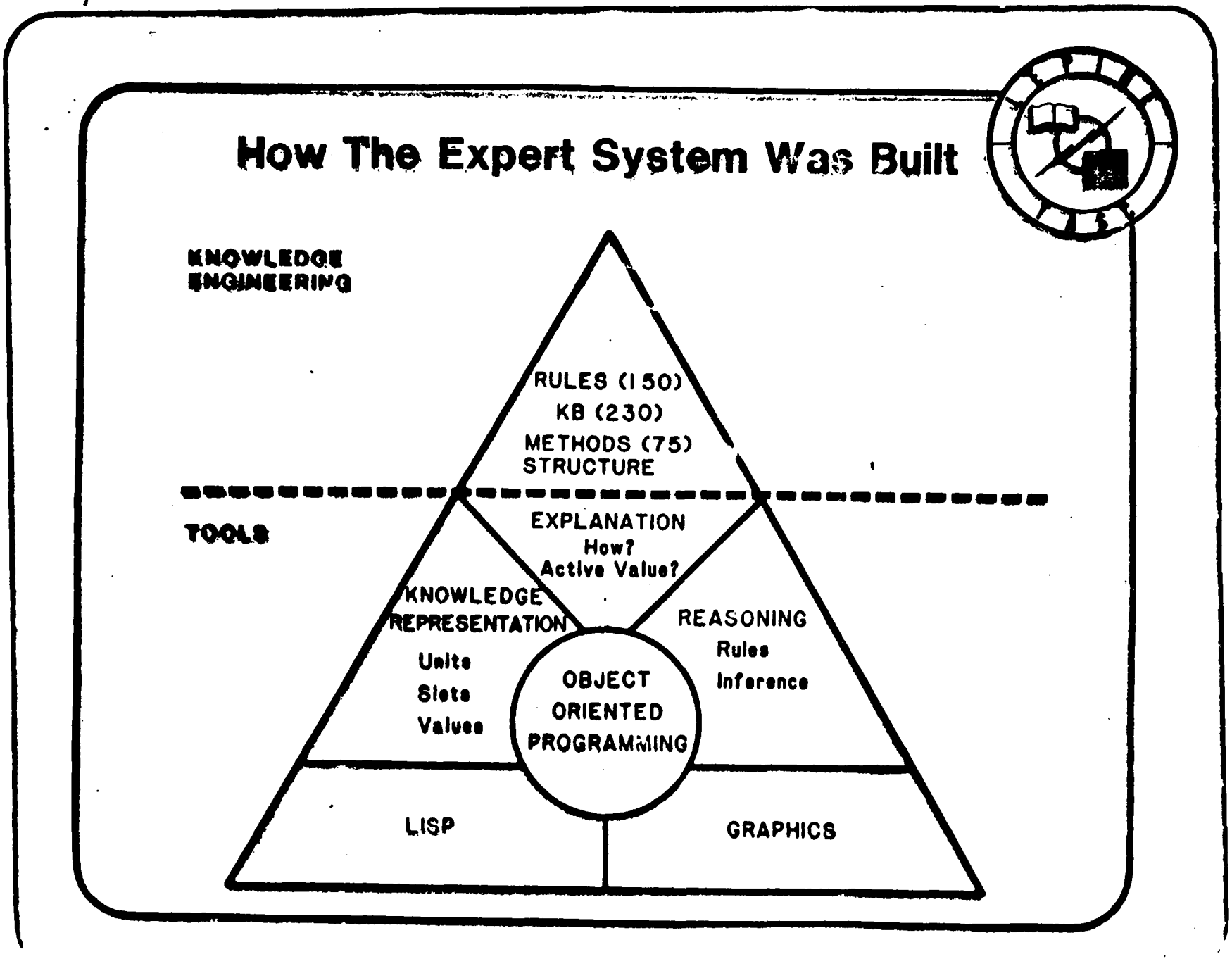
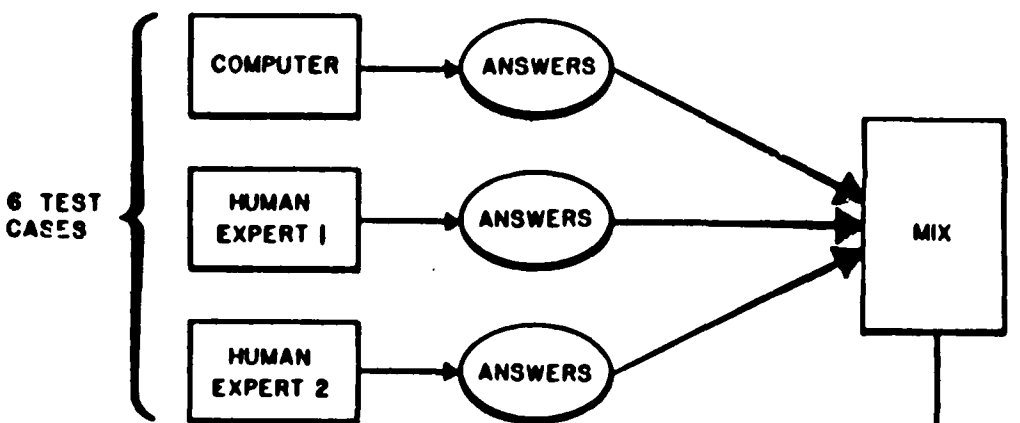

How The Expert System Was Built
KNOWLEDGE ENGINEERING
RULES (150)
KB (230)
METHODS (75)
STRUCTURE
TOOLS
EXPLANATION
How?
Active Value?
KNOWLEDGE REPRESENTATION
Units
Slots
Values
REASONING
Rules
Inference
OBJECT ORIENTED PROGRAMMING
LISP
GRAPHICS

Figure 4.

EVALUATION OF MISSION PLANNER



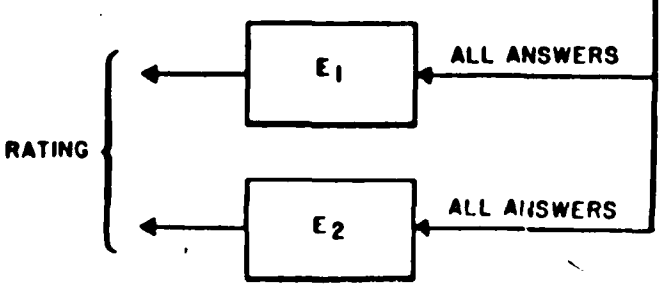
RESULTS:

COMPUTER BETTER-16%

COMPUTER EQUAL-50%

COMPUTER WORSE-32%

IN ALL CASES, COMPUTER ANSWER WAS ACCEPTABLE



Knowledge Base Design and Electronic Troubleshooting

James H. Alexander

AI Department
Computer Research Lab
Tektronix Laboratories
Beaverton, OR 97077

ABSTRACT

In general, my work* has focused upon the development of a knowledge engineering methodology. This has resulted in the conception of Ontological Analysis. On a more specific level, I have been building expert systems for diagnosing failures of electronic instruments. The construction of these systems involved the creation of models simulating the operation of electronic instruments. Practically, the development of these diagnostic expert systems has many implications for the production, and field support of electronic products.

1. Top level research: Ontological Analysis

Today, the design of knowledge bases requires the involvement of an expensive and highly trained knowledge engineer. Typically, the knowledge engineer will analyze a problem, select a problem solving technique, select a knowledge representation, and implement a rule base. This process is guided by the knowledge engineer's experience on an *ad hoc* basis. My colleagues and I have developed a methodology for the analysis of problems. The goal of this research is to develop an articulatable (and teachable) method for creating knowledge-based systems. A paper we are presenting at AAAI-86 proposes a methodology which is an outcome of this work: ontological analysis (Alexander *et al.*, 1986).

The basis of ontological analysis is a principled decomposition of a problem domain. We borrowed from denotational semantics the use of *domain equations* as a tool for describing a knowledge engineering problem. The process begins by identifying in the static ontology of a problem. The static ontology describes the basic elements and relationships between elements in a domain. Second, a dynamic ontology represents the operations that can be performed upon the elements described in the static ontology. Finally, the epistemic ontology contains domain equations describing the methods for selecting appropriate operations to execute at any given time.

Using this high level analysis technique, a knowledge engineer is able to scope out a problem without committing to a knowledge representation or inference scheme. Rather the ontological analysis permits the knowledge engineer to examine the problem in detail and the production of the ontological analysis *guides* the selection of appropriate knowledge engineering methods.

2. Electronic Troubleshooting with expert systems

The Knowledge Engineering group at Tektronix has constructed a number of expert systems for troubleshooting. The troubleshooting project is important for ensuring continued quality product support. Field Service Technicians are increasingly faced with a wide variety instruments to repair, so many that it is difficult to make sure that all technicians are trained to repair

*The work described in this abstract was done in collaboration with the other members of the Knowledge Engineering Group: Mike Freiling, Sherri Shulman, Steve Rehfuss, and Steve Messick.

all instruments. With a set of knowledge based assistants, it will be possible to distribute repair knowledge evenly among technicians. A well designed troubleshooting assistant will make most technicians (but primarily novice technicians) more efficient.

This work has grown out of an early experiment in constructing the FG502-TASP (Troubleshooting ASsistant Prototype; Alexander and Freiling, 1985), which was an expert system for assisting a technician with repairing a simple electronic instrument, the Tektronix FG502 function generator. The FG502-TASP consisted of a knowledge-base sufficient to diagnose 80-90 percent of the most frequent instrument failures. One of the key features of this system was the way it communicated with the end user. The system made strong utilization of graphics such as circuit board pictures, and schematics. The graphics were coordinated so that the user could quickly cross reference parts from a schematic location to a circuit board location. In addition, the system guided the diagnosis by using pointers to indicate precisely where measurements need to be made. The success of this experiment convinced us that strong graphics support is a crucial component to any troubleshooting system.

Subsequent work on troubleshooting focused upon building a general troubleshooting mechanism which could be used to build troubleshooting systems for a whole family of electronic instruments (Alexander *et al.*, 1985). The goal was to build a domain specific inference engine, which would facilitate later construction of troubleshooters. The resulting inference engine (HIPE for Hierarchical Inference Processing Engine, Rehfuss *et al., 1985*) contains a general scheme for trouble shooting called top-down localization. The top-down localization scheme systematically selects tests beginning with the output modules of an instrument. Step by step it exonerates modules and tests modules closer to the inputs. Once a module is deemed bad, the scheme will drop into that module and repeat the procedure at the more detailed level.

Most of the general design decisions for troubleshooting applications are encapsulated in HIPE, thus obviating the need for the knowledge engineer to worry about such decisions when designing a new troubleshooting application. Instead the knowledge engineer can concentrate on problems specific to the current application. HIPE has been used to implement a troubleshooting system for repairing the Tektronix 2236 oscilloscope and is being used in the development of other in-house systems.

One of the continuing topics of interest in this project is the addition of stronger models of the device-under-test to our knowledge base. Currently, the system relies on a very weak model of instrument operation. The level of abstraction used by HIPE presently is one of describing the modules within an instrument, and indicating the causality one module has upon another module. However, in the future we expect to have stronger simulation models of the instrument. The inclusion of knowledge regarding the functionality of each module will allow the expert systems to reason about the operation and interaction of modules, thus creating the possibility of more powerful and accurate troubleshooting schemes.

REFERENCES

Alexander, J.H., M.J. Freiling, S.L. Messick & S. Rehfuss. Efficient Expert System Development through Domain-Specific Tools. *Fifth International Workshop on Expert Systems and their Application,* Agence de l'Informatique Etablissement Public National, Avignon, France, May, 1985.

Alexander, J.H., M.J. Freiling,S. Shulman, J. Staley, S. Rehfuss, and S.L. Messick. Knowledge Level Engineering: Ontological Analysis. In *Proc. AAAI-86*, Philadelphia, August, 1986.

Alexander, J.H. & M.J. Freiling. Smalltalk-80® Aids Troubleshooting System Development. *Systems and Software, 4, 4,* April, 1985.

Rehfuss, S., J.H. Alexander, M.J. Freiling, S.L. Messick & S.J. Shulman. *A Troubleshooting Assistant for the Tektronix 2236 Oscilloscope.* Applied Research Technical Report, CR-85-34; Tektronix, Inc.; Beaverton, OR; September 25, 1985.

Smalltalk-80 is a registered trademark of Xerox corporation.

Simulating Processes over Multiple Abstraction Levels*

Paul A. Fishwick
Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
Moore School D2
Philadelphia, PA 19104

May 21, 1986

Abstract

In dealing with model complexity, most simulation systems have resorted to representing either a restricted part of an overall system or a qualitative model of the entire system. Examples of restricted simulations are *control systems* for electro-mechanical systems (such as helicopter control or control of an engine). Examples of qualitative model simulations can be found in the qualitative reasoning literature[1]. The use of either a restricted or single-level qualitative model is acceptable when an analyst is playing the combined roles of model creator, analyzer, and verifier. But what if an analyst is creating a simulation model that is to be studied and utilized by a arbitrary group of individuals? Some individuals will have a deep knowledge commensurate with that of the model creator. Others may have only a layman's knowledge of the system; they may want to learn about the system by using an exploratory simulation approach.

Some simulation systems have taken a combined *discrete/continuous* approach when studying processes. Examples are systems written using the GASP[4] language and special-purpose combined simulations implemented in an arbitrary general purpose language such as PL/I or FORTRAN. Payne[3] gives a concise overview of combined simulation approaches. The combined approach yields a simulation capability which includes discrete event simulation with the ability to optionally monitor certain variables such as displacements and derivatives in motion equations.

Our approach to simulation promotes the view that complex systems require a better overall organizational approach to studying processes over time. Specifically, we propose a method of simulation which defines a process model over an arbitrary number of abstraction levels. We will exploit the *process abstraction* associated with systems, thereby facilitating a more comprehensible model organization and a great reduction in time complexity due to partitioning. Combined discrete/continuous models can be better represented since it is possible to view some aspects of the discrete model as being abstractions of the continuous model; one is able to separate these abstractions into virtual machines that may execute independently of one another and still communicate via an abstraction bridge (which will be termed *interfacing*).

*This research is supported by NASA Johnson Space Center Contract NAS9-17239, Army Research Office Grant DAAG29-84-K-0061 and NSF CER Grant MCS-82-19196

We have constructed a simulation language termed HIRES[2] (HIerarchical REasoning System) that permits the simulation analyst to create an abstraction hierarchy composed of levels defined (ultimately) using production rule sets. Specific features of HIRES include the ability to control the flow of the simulation either manually or automatically using relations and the ability to design process abstractions using models that are appropriate for those levels. The analyst can construct, for instance, levels in the form of Petri nets, scripts, and continuous block models by using the *pre-processors* that translate these high level descriptions into production rule sets. A slightly modified version of the standard production rule firing algorithm then executes the process. Two example simulations have been constructed to demonstrate the capabilities of HIRES: a four level elevator simulation and a simulation of the "dining philosophers" problem using two levels.

During the course of the research, we have discovered many interesting issues. Abstraction levels, for instance, can be designed using an aggregate technique or via hand-built methods (viz. acquiring domain knowledge from groups or individuals). Both methods are useful; the method employed depends on whether it is desired to 1) maintain consistency among levels, or 2) represent domain knowledge in an effort to *understand* human thought about processes via simulation. Another primary issue in multi-level simulation relates to constructing abstraction bridges between each level. Is there a formalism for constructing a bridge? We have found that interfacing between abstractions seems to be domain dependent — HIRES, therefore, contains a general set of random variates along with the ability to construct heuristics for interfacing. Many issues remain such as analyzing the computational complexity of abstraction levels, and studying the use of entirely different implementation languages or hardware architectures for each level.

References

[1] Daniel G Bobrow. Artificial intelligence. Volume 24: Numbers 1-3, December 1984. Special Volume on Qualitative Reasoning about Physical Systems.

[2] Paul A. Fishwick. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction.* PhD thesis, University of Pennsylvania, 1986. MS-CIS-85-21.

[3] James A. Payne. *Introduction to Simulation: Programming Techniques and Methods of Analysis.* McGraw-Hill, 1982.

[4] A. A. B. Pritsker. *The GASP IV Simulation Language.* Wiley, 1974.

Qualitative Simulation for Estimating Diffusion Transit Times

S. L. Hardt

Department of Computer Science

University at Buffalo SUNY

Buffalo, NY, 14260.

Abstract

Diffusion processes in small but complicated geometries proceed at a high speed and are of great importance for a variety of physical, chemical and biological functions. Exact mathematical formulation of the processes is possible in the form of diffusion equations (partial, second order differential equations) and their appropriate initial and boundary conditions. However, in many cases, it is hard to investigate the behavior of these processes since analytic solutions to the mathematical equations are out of the question and numerical simulations are exceedingly resource consuming.

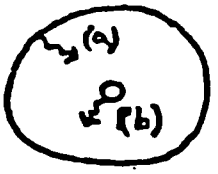
We were confronted with the problem of having to estimate the transit time (mean first passage time) of particles diffusing in small complicated media. We came up with rules of thumb that describe the relations between the pace of diffusion and certain features of the media. These rules are based on the exact solution to the transit time problem (see Hardt 1980b, and Hardt 1984)

The research reported here centers on the qualitative simulation of aspects of the process of diffusion in structured media. In particular, we are interested in the automation of reasoning about the effect that the geometry of the diffusion space and its composition have on the diffusion transit time. We have been

This research is supported by the National Science Foundation under grant number MCS 8305249.

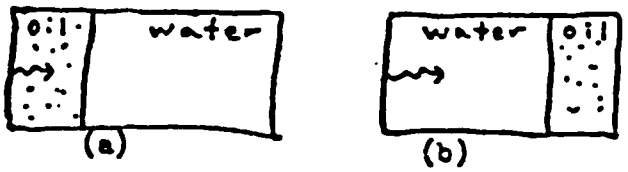
developing the logic required for the representation of the knowledge used to reason about the pace of diffusional processes. In addition, a theory is developed to answer the following two questions: (1) How to coordinate the use of "deep" and "shallow" causal knowledge, and (2) how to determine when to stop the reasoning process. To demonstrate the issues involved in our research, consider the following two problems:

Molecules released from one spherical surface diffuse to, and are trapped by a second, concentric, spherical surface. The radius of the inner surface is much less than the radius of the outer surface. Case (a): the molecules are released from the outer surface and diffuse inward. Case (b): the molecules are released from the inner surface and diffuse outward. Compare the transit times.



Longer in (a) [ ]
The same in both [ ]
Longer in (b) [ ]

Molecules released from the surface on the left diffuse to, and are trapped by the surface on the right. The molecules are slightly soluble in water, and highly soluble in oil. In case (a) they diffuse through the layer of oil before reaching the layer of water. In case (b) the thickness of each layer is unchanged but the order is reversed so the molecules diffuse first through the water and then the oil. Compare the transit times.



The same in both [ ]
Much longer in (a) [ ]
Much longer in (b) [ ]

Using common sense knowledge based on experience with everyday-life physical events, may lead to the following reasoning: since in both cases (a) and (b) the diffusing particles travel the same total distance, and are exposed to the same overall medium and volume, the transit time for both cases should be the same. Although this above reasoning, which was based on intuition about inertia processes produced the wrong answer, it may serve as a window into the knowledge people have about every-day life physics. In partic-

ular, the knowledge used to produce this reasoning is process-based (Forbus 1984) since it centers on the flow process. Hence, the above reasoning pattern suggests that the process description of flow contains knowledge about distance, volume and total media and their effects on transit times for flow processes.

In order to correctly and effectively estimate relative values of transit times in complicated structures, the following two expert hueristics which can be based on the formal theory of Brownian Movements or the theory of second order partial differential equations. (Hardt 1979 and 1980a,b) should be considered:

(1) Movements of individual particles are mutually independent and independent in each degree of freedom.

(2) The accessibility of a target is determined by: (a) The *size* (largest linear dimension) of the target. (b) The *affinity* of the diffusing particles to the immediate target environment.

For the quiz problems presented above, it is enough to use the second hueristics and to realize that the transit time is very strongly dependent on the accessibility of the target. Hence, in these examples, accessibility is a central concept and it drives the reasoning.

There is a great similarity between building a simulation model and building a knowledge base. When mathematics is applied to problems in the natural world, the resulting formulation is rich with the mathematician intuition. The thought processes that resulted in the formulation are, in an important sense, incorporated in it (e.g. Lin and Segel 1974). Artificial Intelligence (AI) approaches to the problem of qualitative reasoning may be placed along the conceptual line connecting the original intuition and the resulting mathematical equations (when they exist) in the above scenario. At the one extreme on this line, AI provides an approach that can be viewed as reasoning about simplified qualitative (mathematical) equations. (see De Kleer and Brown 1984, Kuipers 1984). At the second extreme, AI provides an approach that can be viewed as reasoning about processes with no special emphasis on causality (see Forbus 1984). A system using the first approach to solve the quiz problems reasons about a qualitative version of the difference equations used to

numerically integrate the diffusion equation for this case. A system using the second approach, reasons about the process of flow in structured channels, with special emphasis on the effect of channel geometry and composition on the flow rate and the channel hold-up. Both these approachs can be augmented into a numerical simulator to provide more effective resource utilization.

To implement and further investigate the process of qualitative reasoning about diffusion transit times, we have developed the DUNE (Diagnostic Understanding of Natural Events) system architecture that organizes the knowledge around concurrent processing structures. The system was designed as a shell for expert-systems that aid diagnoses assessment and problem solving tasks (see Hardt et al 1986). The full presentation will include the formalism that support the reasoning as well as a discussion of the computer implementation.

Acknowledgement. This research is supported by the National Science Foundation under grant number MCS-8305249.

References.

De Kleer J. and Brown J.S. (1984). A Qualitative Physics Based on Confluences. *Artificial Intelligence*, **24**.

Forbus K.D. (1984). Qualitative Process Theory. *Artificial Intelligence*, **24**.

Hardt S.L. (1979). Pace of Diffusion Through Membranes. *Journal of Membrane Biology*, **48**, 299.

Hardt S.L. (1980a). The Diffusion Transit Time: A Simple Derivation. *Bull. Mathematical Biology*. **87**, 1.

Hardt S.L. (1980b). Diffusion in Structured Media. Ph.D. Thesis, The Weizmann Institute of Science.

Hardt S.L. (1984). Naive physics and the physics of diffusion. Or: when Intuition fails. Research Report number 211, Department of Computer Science University at Buffalo.

Hardt, S.L. MacFadden, D. Johnson, M., Thomas, T., Wroblewski, S. (1986). *The DUNE Shell Manual: Version 1.* Research Report 86-12, Department of Computer Science, State University of New York at Buffalo.

Kuipers B. (1984). Common Sense Reasoning About Causality: Deriving Behavior from Structure. *Artificial Intelligence*, **24**.

Lin C.C. and Segel L.A. (1974) *Mathematics Applied to Deterministic Problems in the Natural Sciences* Macmillan Publishing Company, New York.

Artificial Intelligence and Simulation

Robert O'Keefe*, John Roach and Renga Iyengar
Department of Computer Science,
Virginia Tech,
Blacksburg, VA 24061.

May 20, 1986

1 Introduction

Research on combining AI and discrete simulation is being pursued in the Department of Computer Science at Virginia Tech. Research is concentrating on both developing simulation tools that are usable within AI systems, particularly in the area of robot planning, and integrating AI capabilites within conventional discrete simulation frameworks.

2 The Prolog Simulation System (PROSS)

One of the major failings of discrete simulation, and existing Simulation Programming Languages, is the inability to model intelligent behavior. Frequently the discrete simulation developer makes assumptions about the system being modelled (for instance, the assumption of non-adaptive behaviour by all objects within the simulation) so as to avoid having to model *intelligence*.

To overcome this, by providing a tool that allows such behavior to be modelled within conventional discrete simulations, PROSS (The Prolog Simulation System) (1) has been developed. PROSS is an implementation of the General Purpose Simulation System (GPSS) in Prolog. Within description of object processes, Prolog goals can be freely mixed with GPSS block statements. Thus at one extreme a PROSS process description can be a conventional GPSS process description; at another extreme it can be an agenda of goals. Process descriptions in PROSS can be dynamically altered within the running simulation, and thus models of expert reasoning can be used to alter or plan object actions as the simulation proceeds.

*on leave from the University of Kent at Canterbury, England

3 Using Simulation in Multiple Robot Planning

An initial multiple robot planning and execution system has been developed (2). This system uses multiple operating system processes to simulate parallelism. This approach proved inflexible, and consumed many resources, so a discrete simulation system has been constructed to model interaction in multiple robot systems. The system allows robot plans to be tested on hypothetical worlds. At present, all work is proceeding with Prolog. Ultimately, a generalised multiple robot simulation tool will be implemented in PROSS.

4 Machine Learning in the Development of Discrete Simulations

One of the hardest parts of simulation modeling is the production of a programmed model from a conceptual model. This task could be aided, or even circumvented, if software can be constructed that in some sense *learns* about the system to be modelled, and automatically constructs all or part of the simulation model.

As a start on the investigation of machine learning to this problem, Quinlan's induction algorithm ID3 has been used to induce the necessary conditional events in a discrete-event simulation model from a set of examples (3). The experience was encouraging, in that valid conditional events were produced in a form suitable for future coding in a Simulation Programming Language. However, ID3 is very constraining - a valid event can only be guaranteed if a complete set of examples can be provided.

Ultimately, research efforts in machine learning other than induction may be more applicable. For instance, *learning-by-analogy* could be applied to discrete simulation, with the intention of producing a system that can generate a simulation model for domain a given a simulation model for domain b, and the characteristics or constraints that make a differ from b.

5 Advisory Systems for Discrete Simulation Users

Increasingly, discrete simulations are being used by the model sponsors, rather than the model developers. Advisory expert systems hold the promise of being able to provide expertise on the use of simulation models to naive users. An expert system that helps users determine appropriate experiments with transaction-flow discrete simulation models has been developed (4). Called TRANS, it has been constructed using the ES/P Advisor shell and Prolog.

A number of conclusions have resulted from the development of TRANS. Firstly, the inadequacy of a rule-based approach, and the need to represent simple causal relationships (for instance, increasing resource provision may result in decreased queue lengths) within the system. Secondly, the problem of terminology. Simulation analysts use a fairly precise

meta-language for discussing simulation models. TRANS employs this language. However, model sponsors typically want to talk about a model in terms of their own world. Thus any general advisory system may need to be *tailored* to any particular application area.

Due to these problems, work on TRANS has been suspended. Any future attempt at producing an advisory system for simulation will have to address the issue of how best to represent simulation knowledge, and combine it with knowledge about the application domain.

References

1. R. O'Keefe, J. Roach and R. Iyengar, "PROSS: adding *intelligence* to discrete simulation using Prolog", In preperation.

2. J. Roach and P. Montague, "Coordinating multiple robot arms to increase productivity", in *Computer-Based Automation (J. Tou, ed.)*, Plenum, New York, 1985.

3. R. O'Keefe, "Simulation and expert systems - a taxonomy and some examples", *Simulation* 46 (1986), 10-16.

4. R. O'Keefe, "Advisory systems in simulation", in *Artificial Intelligence Applied to Simulation (Kerchoffs, Vansteenkiste and Zeigler, eds.)*, The Society for Computer Simulation, La Jolla, 1986.

Lawrence E. Widman, M.D. [23]
Division of Cardiology
Department of Medicine
University Hospitals of Cleveland
2065 Adelbert Road
Cleveland, Ohio 44106

REASONING ABOUT DIAGNOSIS AND TREATMENT IN A CAUSAL TIME-VARYING DOMAIN USING SEMI-QUANTITATIVE SIMULATION AND INFERENCE

Yong-Bok Lee* and Lawrence E. Widman, MD, PhD*+

*Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland OH; +Division of Cardiology, Case Western Reserve University; and +Clinical Decision Making Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA.

Keywords: scenario construction for expert systems, integration of AI techniques with conventional simulations, knowledge representation formalisms for simulation.

This project seeks to develop improved knowledge representation and reasoning algorithms for domains dealing with time-varying phenomena. Such domains in medicine include cardiovascular hemodynamics; renal pathophysiology; endocrine, biochemical and genetic pathways; and respiratory management. Improvements in these algorithms could allow expert programs to (1) form differential diagnoses on the basis of available signs and symptoms, (2) recommend treatment based on a global understanding of the given patient's disorder, (3) evaluate the outcome of treatment to refine the initial diagnosis and to detect new concurrent diseases, and (4) interact with the user on a symbolic level when explaining its reasoning and acquiring new information. Such algorithms could also be applied to other time-varying domains.

The fundamental approach consists of defining functional *building blocks* for describing the expert domain in symbolic terms, translating the domain into a set of first order differential equations on the basis of the building block definitions, establishing initial conditions for the equations on the basis of signs and symptoms, and integrating the equations numerically by standard techniques. This approach combines the descriptive power of mathematics with the symbolic reasoning power of expert system technology.

The use of numerical integration of differential equations for physiological modeling has been well developed by Guyton and others. With the judicious use of default values and semi-quantitative arithmetic definitions, the inexact information available to the expert program is sufficient to permit satisfactory simulation. Feature extraction on the simulation output translates the results back into the symbolic domain, allowing the program to make inferences regarding historical and future events on the basis of the known causal relationships of the domain and its knowledge of the patient.

The approach has been tested in the cardiovascular domain. A symbolic model was built using only well-established relationships. To test the default capability of the approach, very few of the relationships were specified quantitatively: most were specified as links consisting of causative node, affected node,

and direction of influence (positive or negative). Time delays were specified by order of magnitude. The program was written in Maclisp.

The performance of the model was tested on sixteen classic cardiovascular disorders, such as decreased contractility, increased pulse rate, decreased systemic vascular resistance, and decreased gravity. In all cases, the output of the simulation was qualitatively correct: all variables changed in the appropriate direction on the proper time scale with semi-qualitatively correct magnitudes of change.

The next step, establishing the initial conditions for the equations on the basis of signs and symptoms, is now in progress. It consists of searching among all possible states of the model for those which include the given signs and symptoms. Signs and symptoms are translated into semi-quantitative values in an ordered quantity space. Each state of the model is a causally consistent specification of one value for each variable in the model. Each state is also a diagnosis, since deviations from normality in the model imply the existence of one or more primary faults or derangements (diagnoses) and compensatory secondary changes.

The basic approach is depth-first search combined with semi- quantitative constraint propagation. The directed graph representing the physiological model is converted to a tree by terminating cyclic linkages. The tree is traversed bottom-up and top-down as permitted by the given signs and symptoms. Search *paths* consist of the tentative *values* which are assigned to variables as the tree is traversed. New values are constrained by values previously assigned to causally related variables considered elsewhere in the tree. Creation of new search paths occurs when more than one semi- quantitative value can be assigned to a variable under the available constraints. Domain-independent heuristics limit the search space by forbidding new values leading to physically impossible situations.

Unlike ordinary tree searches, all branches in this tree are considered in turn for each search path unless the path is abandoned. Paths are abandoned when inconsistent constraints are identified in cyclic links in the model or in variables for which given values are available from the signs and symptoms.

Possible diagnoses are found by matching value assignments against a list of diagnoses and their primary derangements. Currently, diagnoses are limited to exactly one derangement. Preliminary results show that this algorithm identifies correctly all single-fault diagnoses consistent with a given set of signs and symptoms. Issues of the relationship of physiological thresholds to the quantity space, identification of multiple diagnoses and of diagnoses with multiple faults, and the match between value assignments and the corresponding simulation output require further investigation.

Symbolic model-based simulation using functional building blocks for describing the expert domain in symbolic terms may be useful in a variety of physically realizable domains. The symbolic form of the model also lends itself to simulation at multiple levels of abstraction and automatic analysis of simulation results.

Steven C. Bankes, Ph.D.
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406
(213)393-0411
arpa-net: bankes@rand-unix

There are a variety of projects at the Rand Corporation which aim at the hybridization of knowledge-based and simulation technologies. These range from research in fundamental issues to the development of large applications for delivery to end users. I am involved with a number of these projects in roles ranging from kibitzer to principle investigator.

RAND Strategy Assessment System

I am project leader on the National Command Level modeling project for the Rand Strategy Assessment Center (RSAC). The RSAC is a large ($4-5 million/year) program developing a software system called the Rand Strategy Assessment System (RSAS). A beta-test version of this system will be delivered within the next month. The RSAS is a very large simulation and (man-machine) gaming facility which incorporates both traditional simulation and knowledge-based models. The knowledge-based portion includes both rule-based and script-like models. The RSAS must certainly be one of the largest existing simulations using AI techniques. I am one of the designers and implementors of the RSAS as a whole, and I also have primary responsibility for the National Command Level (NCL) models in the system.

The National Command Level models simulate the decision making of national political leadership during large scale global crises. When awakened, these models make a series of reasoning steps resulting in decisions about global and operational objectives and strategies. We are developing a number of these models, with different versions for various hypothetical types of leadership for each of the superpowers. The NCL models serve as components of the overall simulation and also stand independently as important tools for research in political science.

One of the unique features of these models is their use of a simulation as a knowledge source. In fact, by virtue of a technique we are calling recursive simulation, the NCL models can run a version of the overall RSAS to assist in decisionmaking. The models specify assumptions about the opponent, the nature of other actors, and the laws of causality of the game world. These assumptions are used in performing a "look-ahead" simulation. Thus, the rule-based models can use the simulation to ask a variety of what-if questions. This has proven to be a powerful method. Because these assumptions can be updated as a result of game history, we are able to model phenomena of learning during course of a conflict, as well as deception and feints aimed at producing false learning in one's opponent.

Knowledge Based Simulation

I am also involved in the Knowledge Based Simulation Project. (For a full description of this effort see Jeff Rothenberg's abstract, also being sent to you.) My interests here are in a more general exploration of using simulations as knowledge sources for rule-based systems, and in a complex of ideas I call "chunking with objects".

Based on experiences with the RSAC I have the belief that using simulations as knowledge sources can be a powerful technique (and one which is not often included in lists of ways that artificial intelligence and simulation can be usefully combined). In order for this technique to be applied broadly, some problems must be addressed. General methods to allow rule-based systems to specify particular simulations to be run need to be developed. The variety of ways in which the results of such simulations could be exploited by knowledge-based systems needs to be explored. We also need to better understand the trade-off between representing knowledge as a simulation model as opposed to explicit representation as rules or frames. Essentially, the simulation models serve as representation for certain kinds of deep knowledge, with rules representing more shallow knowledge structures.

Object-oriented methodologies have provided a major technical basis for combining AI techniques with simulation. Our experience with ROSS and other object-oriented languages have led us to a number of observations about the limitations of such tools. Particularly compelling to me is the difficulty in building very large systems using object-oriented methods. Much of this difficulty results from the lack of suitable constructs to properly "chunk" the knowledge which is represented in the system. A generalized version of the object-oriented paradigm could provide a number of features which would assist both the programmer and the end user. We are working on methods for allowing a variety of self-contained local points of view in both the coding and execution of these systems. Our approach includes not only scoping facilities, but ideas like local simulation, multiple presentation perspectives of the simulated world, and representation of multiple levels of aggregation.

"Orthogonal" Views of a Simulation

Dr. Malcolm R. Railey [28]
The BDM Corporation
7915 Jones Branch Drive
McLean, VA 22102-3396

At BDM Corporation, we are developing an expert system to integrate the tasks of planning, control, and analysis of a complex (i.e. global or integrated) simulation. An integrated simulation represents a combination of one or more low-level or primitive "entities". Each entity perceives the combination of all other entities as the "environment" with which it may react. Each entity also possesses a set of "characteristics" that express intra-properties such as its operation and attributes, and inter-properties such as its input/output requirements and its causal relationships with its environment.

The planning of a simulation involves the specification and definition of all entities and their environments. In controlling a simulation, one not only executes the integrated simulation, one also alters or perturbs the behavior of one or more entities and their environments. Controlling an entity's environment enables one to observe and therefore model certain behaviors of the entity. The objectives of analysis determine the observable variables (the entities to be observed) in the simulation, and thus they determine the control variables as well. The components of an entity's environment must be controllable, hence these components must be defined at the necessary level of abstraction (i.e. granularity) to be controlled. The objectives of analysis affect the plan and control of the simulation. During analysis, one observes the effect

of the perturbations on the behavior of different entities, and then determines whether the objectives of analysis are achieved.

Our goal is to achieve a "complete" and "correct", "controllable", and "observable" simulation. First and foremost, an integrated simulation must be complete and correct. By complete, we mean all entities that must be integrated in the simulation are integrated. By correct, we mean the entities are combined in a manner consistent with their inter-properties. Without correct specification of the observable and control variables, one may not focus on particular behaviors or attributes of the integrated simulation, and thus an analysis of the integrated simulation with respect to the objectives of analysis is not possible. The key to planning a controllable and observable simulation is in the statement of the objectives of analysis, and in how this statement is incorporated into the planning and controlling phases of the simulation. Our approach here is to incorporate "orthogonal" views of a simulation in order to integrate these three tasks.

"Orthogonal" views of a simulation are not simply multiple levels of abstractions, but rather totally different views of the same simulation. For example, suppose we wish to determine the effectiveness of an automobile as a means of transportation. Using multiple levels of abstraction, we may model the operation of and interaction between an automobile, its driver, and its environment. From an orthogonal view, the automobile is modelled as a means of transportation rather than a mechanical device, the driver is modelled as a person that uses the automobile as transportation rather than as the operator of the vehicle, and the environment is modelled not as the road, traffic, or weather, but instead the

perception of the person towards his means of transportation in regards to its effectiveness. During planning, one may completely and correctly specify all entities necessary to describe the vehicle, the driver, and the environment. Incorporating the orthogonal view (the automobile, driver, and the driver's perspective towards efficient means of transporation) guarantees that the necessary observable and controllable variables in the integrated simulation are present.

To completely and correctly specify an integrated simulation, we consider different methods of describing the entities and their environment at various levels of granularity. We then consider how to incorporate orthogonal views into the integrated simulation so as to guarantee the observability and controllability of the simulation.

A Knowledge-Based Simulation Model for Evaluating Alternative Courses of Action

Ina Ghaznavi [30]
Strategic Systems Division
GTE Government Systems Corporation
1 Research Drive
Westborough, MA 01581

The military community is continually faced with making timely decisions that involve uncertainties, incomplete data, and alternative courses of action. The problem solving process involves collecting relevant information to effectively gauge several feasible courses of action in achieving desired objectives and then selecting the best option.

Conventional simulations for modeling and evaluating alterative courses of action have been limited in performance and use. This has been attributed to: a) the programming effort required to build and alter the models, b) the models' reliance on numerical problem-solving techniques to represent and explain behavior that may not be amenable to numeric computations, and c) the laborious task of gathering and analyzing simulation data to interpret the results and evaluate the sensitivity of each option to "what-if" situations.

A Crisis Action Planning Model (CAPMOD), has been developed to demonstrate a "knowledge-based simulation" approach to enhance existing military planning models. This decision support environment integrates a discrete simulation with a knowledge-based system and object-oriented graphics. To date, prototype demonstrations have verified its ease of use in setting up and modifying course of action simulations and improved user understanding of simulation results.

The generic model simulates force movement flow processes from some point of origin to an objective area. It models a military movement network that is comprised of military bases, routes that connect bases, movement assets to move force units between bases, and military assets to deploy. The modeling approach is a discrete simulation based on queuing theory.

The evaluation of courses of action is complicated by such diverse, conflicting and inter-related planning factors as resource requirements and availability, terrain and weather restrictions, the disposition of enemy forces, and competition for resources due to concurrent missions. Therefore, modeling the effects of situational and exogenous factors on the deployment of military assets is essential for accurate simulation. They must be represented in the simulation and applied in reasoning about the acceptability or shortfall of a course of action.

The current model simulates delivery delays and attrition of forces due to situational and exogenous factors. The effects of these factors (eg. terrain, weather, and enemy action) on force deployment are actually expressed in rules in a knowledge base. Rule representation was chosen because they are used by

military planners in describing force deployment behavior and they facilitate human understanding of cause-effect situations.

The production rules are invoked during the simulation process and when explanation of simulation results is required. It is insufficient to just compute a measure of effectiveness (eg. the difference between the planned and actual accumulation of forces at an objective area). The user needs to understand "why, when, and where" the plan failed so recommendations for corrective replanning can be derived and supported. This level of explanation is achieved by CAPMOD by applying rules for hypothesizing about the potential causes of inconsistencies or variance in a plan to meet an objective. In-depth explanation is provided by backtracking through a force deployment plan and the rules that derive the simulation solution. The purpose of backtracking through a course of action model is to selectively collect relevant data that isolate the cause(s) of variances in a plan.

The knowledge base also provides a symbolic representation of the objects, relations, and behavior of the military organization that can be represented in the model. The basic element for representing the objects is the unit frame. This simplifies the use of the model and maximizes user understanding of the simulation results because the concepts presented are familiar to the end user. An object-oriented user system interface, which provides a graphical representation of generic model objects linked to the knowledge base, further assists the user in model building.

To easily migrate from one course of action model to another, the knowledge base provides a library of generic model objects (ie. military assets, their attributes and relations) essential for representing a force movement network. Scenarios are created by modifying attribute values (through the user and relevant data bases). This provides a vehicle for the user to initialize and alter the model entities and behavior, particular to a planning problem without altering the generic model.

The formulation and evaluation of a course of action can be viewed as a constraint-directed activity, where constraints serve to discriminate and restrict hypotheses generated. Constraint-directed reasoning is applied in validating user inputs in creating the course of action model. For example, the user may specify the number of forces to be deployed from a base of origin, but if another mission objective has been defined that demands these resources, the system will alert the user of a conflict. Types of constraints represented are resource requirements and availability.

Finally, a blackboard keeps track of current and past simulation events of courses of action modeled. It provides the means to process and compare multiple alternatives in interim simulation runs.

Future work will continue to focus on explanation, through the application of constraint-directed reasoning to support the comparison of multiple simulation runs. The CAPMOD system will be integrated with color graphics to augment the user system interface.

SIMSMART:
DYNAMIC SIMULATION FOR ENGINEERING DESIGN, OPERATOR TRAINING AND AUTOMATED CONTROL OF INDUSTRIAL PROCESSES

The secure operation of an industrial process depends on the human operator and the real-time process control system. Both of them must perform well together. The operator, in order to make decisions; depends upon the availability, the timeliness and the quality of the information presented to him by the process control system. If the control system does not perform well, the operator, no matter how capable he might be, would have great difficulty in making the right decision. On the other hand, even if the control system does perform well, the operator may not be psychologically or technically prepared to make the right decision.

Due to the infrequency of occurrance of upset conditions in the process, the ability of the operator and the control system to cope with these situations remains untested. It is therefore quite clear that in order to efficiently "train" the operator and also "program" the control system, it will be necessary to use simulation of normal and abnormal operating conditions, both in real-time and faster-than-real-time situations.

The subject of this paper is the concept of a fifth generation simulator called **"SIMSMART"**, combining well-proven techniques in process/process control simulation, computer-aided-design (CAD), man-machine interface and data-base management. Some features of the **SIMSMART** simulator are:

1° Realistic integrated dynamic simulation of the process equipment and process control systems in real-time and faster-than-real-time.

2° Realistic emulation of the process control system including control algorithms, configurations, graphics, etc... based on manufacturer's data.

3° Realistic emulation of the plant equipment; including pumps, piping, control valves, tanks, etc... based on manufacturer's data.

4° A computer-aided design (CAD) processor for the non-technical man-machine interface. The process flowsheets (P and ID) are reconstructed using ICONs and menus in a conversational manner.

5° All levels of simulation can be examined due to the "ZOOM" feature; from the minute details of a PID loop to the overall operation of a unit process.

6° The on-line simulator comes equipped with an automatic model building tool based on process measurements, statistical analysis and parameter estimation techniques.

SIMSMART has been designed for all stages of a project, from a machine rebuild to a greenfield mill installation; starting from the design engineering phase through to plant start-up and commissioning.

Don Waye [31]
Applied High Technology Ltd.
2, Place Yvon Plourde, Suite 205
Charlemagne (Quebec)
J5Z 3E8
CANADA

SIMSMART begins with flow sheet balancing, process modelling and process control design and simulation studies.

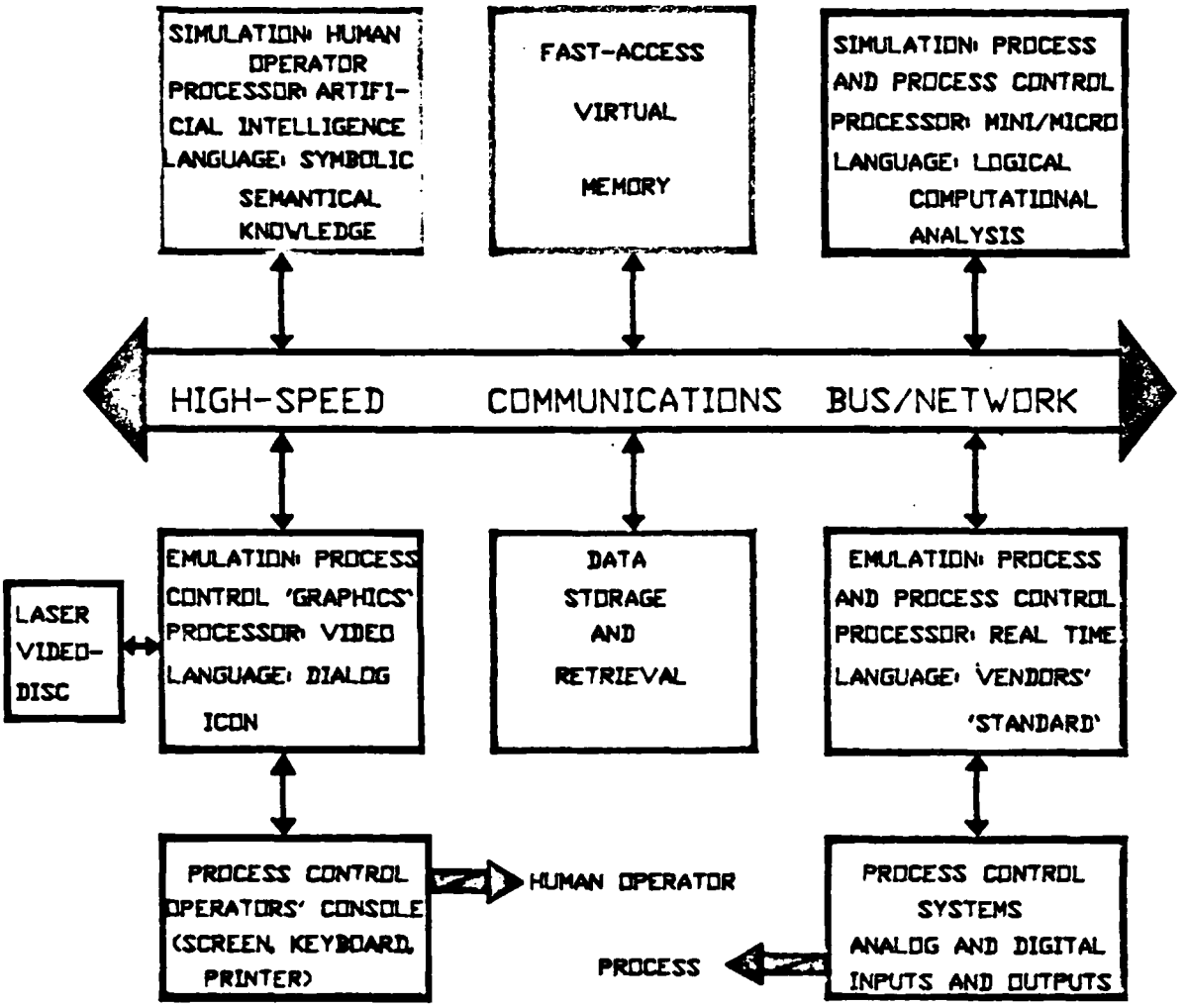
SIMSMART becomes an integral part of operations personnel training through frequent "What If" sessions for process start-ups, shut-downs, production rate changes, grade changes, etc...

SIMSMART monitors on-line measurements during start-up and commissioning and can build and refine process models with the automatic model builder.

SIMSMART updates automatically the process control emulation depending on the configuration of the control system when any changes are made to either the analog control system (distributed controls) or the discrete control system (programmable controllers).

SIMSMART's automatic features such as the process model builder and the control system emulators provide a simulator whose high degree of reality will not degrade with the inevitable ongoing modifications to both the process equipment and the process control systems.

SIMSMART



HTA Ltee

SYNTHETIC INTELLIGENCE FOR PROCESS, PROCESS CONTROL AND HUMAN OPERATOR SIMULATION ENGINEERING DESIGN, OFF-LINE OPERATORS' TRAINING AND ON-LINE PROCESS CONTROL, PROCESS OPTIMISATION AND MILLWIDE CONTROL

Abstract for Engineering Workshop AAAI 86
Workshop on AI and Simulation

Richard M. Adler, Ph.D.
The MITRE Corporation
Bedford, MA 01730
rma@MITRE-BEDFORD

A serious need exists today for increased automation in the Government's systems acquisition process. Automation refers to software utilities supporting the generation and analysis of system design descriptions. Generation encompasses design description creation and modification. Analysis encompasses behavioral modeling and formal methods for verifying design completeness, correctness, consistency, and feasibility. I am directing a project aimed at constructing a knowledge-based environment for developing such tools. The initial prototype is being implemented on top of KEE, running in a Symbolics environment. Design descriptions for a military communications system are being used as application test vehicles.

The basic architectural principles underlying the project are as follows:

- Design descriptions are generated from a modeling library, using a simple copy-and-edit strategy to specialize model element templates to particular designs. The library contains frames that characterize prototypical C3I components (e.g., sensors, archetypal network configurations, platforms, processors, operating systems).

- The user interface is highly visual in nature. Primary building blocks include multiple windows, pop-up menus, and, most importantly, icons representing system design components (viz., hardware, software, and data objects). Icons can be browsed to different levels of detail: an option offered to a user mousing an icon depicting a component as a discrete entity, is to invoke a more detailed window depicting that element's subcomponents and their configurational structure.

- The functionality of the system is represented in two ways: passively, via descriptive frames, and actively, through behavioral simulations. The simulations rehearse the event sequences in the model hardware-software configuration that implement system functions. Basically, the frames representing components and data objects have state

slots, whose contents are updated as system processes transpire. The simulation model is represented graphically through icon animation sequences.

The initial stage of the project concentrates on qualitative, nonexhaustive models of design descriptions. That is, KINDS of functions, components, behaviors are modeled, rather than all functions, components, and behaviors. For example, message-parsing is depicted as a general parsing capability, rather than as a complete (message-specific) field-by-field representation. Similarly, host processor functionality is modeled in terms of multiple workstations (i.e., several) rather than the full complement that the complete requirements specifications call out. Models in later stages of the project will simulate more detailed system descriptions, in order to support exact, quantitative performance analyses.

The attached diagram illustrates the architecture for the behavioral model of a message-handling system. The level of detail is one of a high-level functional specification. The model is constructed so as to minimize presuppositions or commitment to particular design implementations: functional specifications are deliberately written in this fashion in order to leave contractors as free as possible to devise their own architectural strategies. Another behavioral model, in the early stages of development, simulates a preliminary design of the message-handling system that stipulates a specific architectural approach. Accordingly, this more detailed model depicts explicit structures and behaviors reflecting the posited system configuration and mechanisms.

The simulator is run by (object-oriented style) methods, activated by messages sent out from units (i.e., frames), representing a system clock, the message handler, and specific functions. The attached screen dump portrays the functions characterizing the message-handling system organized in a taxonomical hierarchy. The functionality of the message handler is encoded in two separate knowledge bases, represented in terms of rules driven from a forward-chaining inferencing strategy. One knowledge base identifies the message with the highest urgency for processing, based on time of arrival, designated message priority level (assigned by the simulator), and current processing state of the message. The second knowledge base determines the appropriate processing for the highest urgency message.

An attractive feature of this model architecture is its extensibility. The simulation can easily be upgraded so that the clock polls an operating system unit on each cycle. The latter unit then activates not only the AMH, but other applications hosted by the central processor as well (e.g., a message-handling statistics

utility). This extension could be implemented by revising the clock polling method, adding an operating system frame, and by constructing a knowledge base describing operating system control functionality: the models for the message-handling system, and all other applications, determine their own activities through reasoning based on purely internal structures (viz., knowledge bases). The only nontrivial modification to implement the extension is the development of the control knowledge base, which can be derived from the operating system's behavioral (i.e., declarative) functional description.

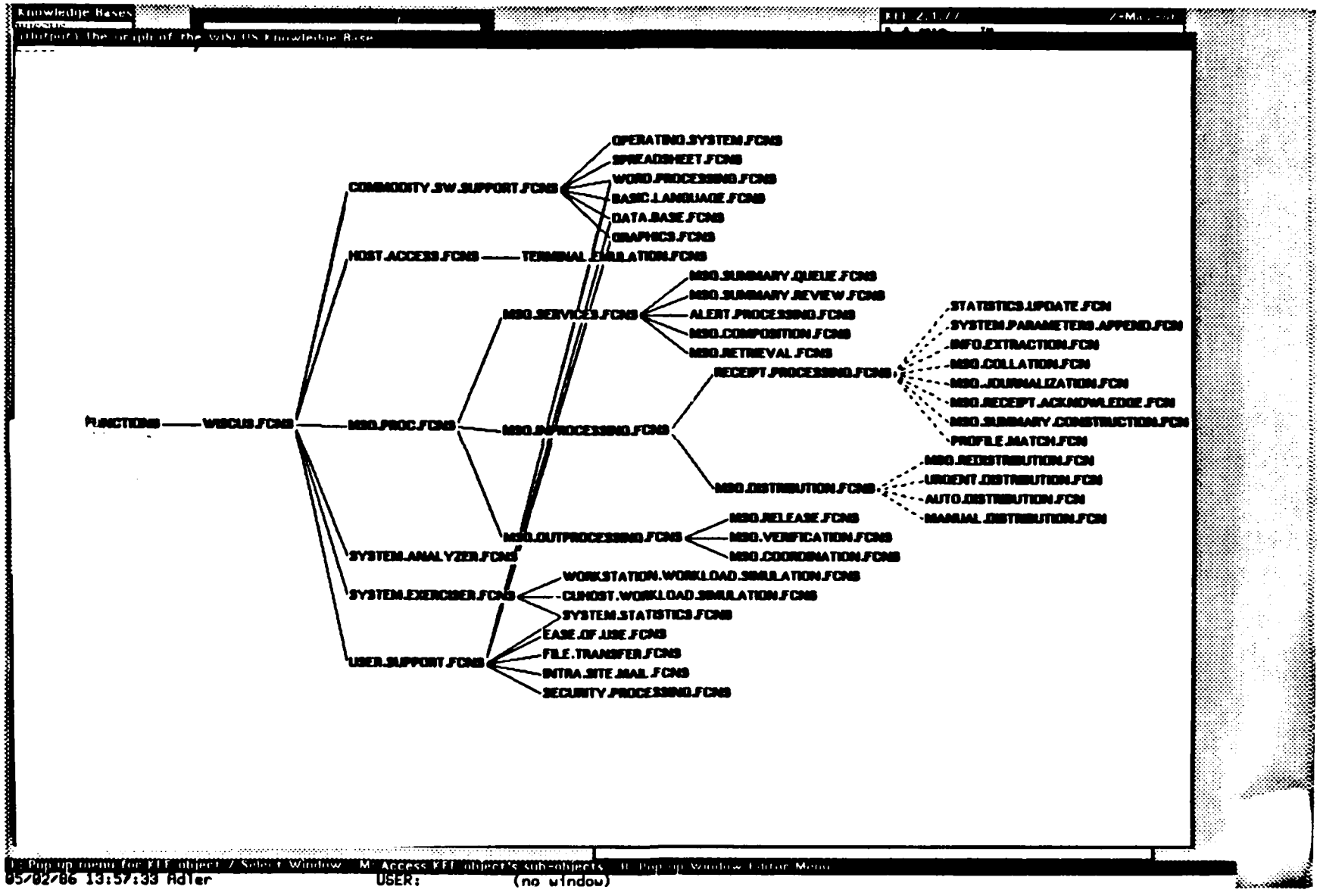
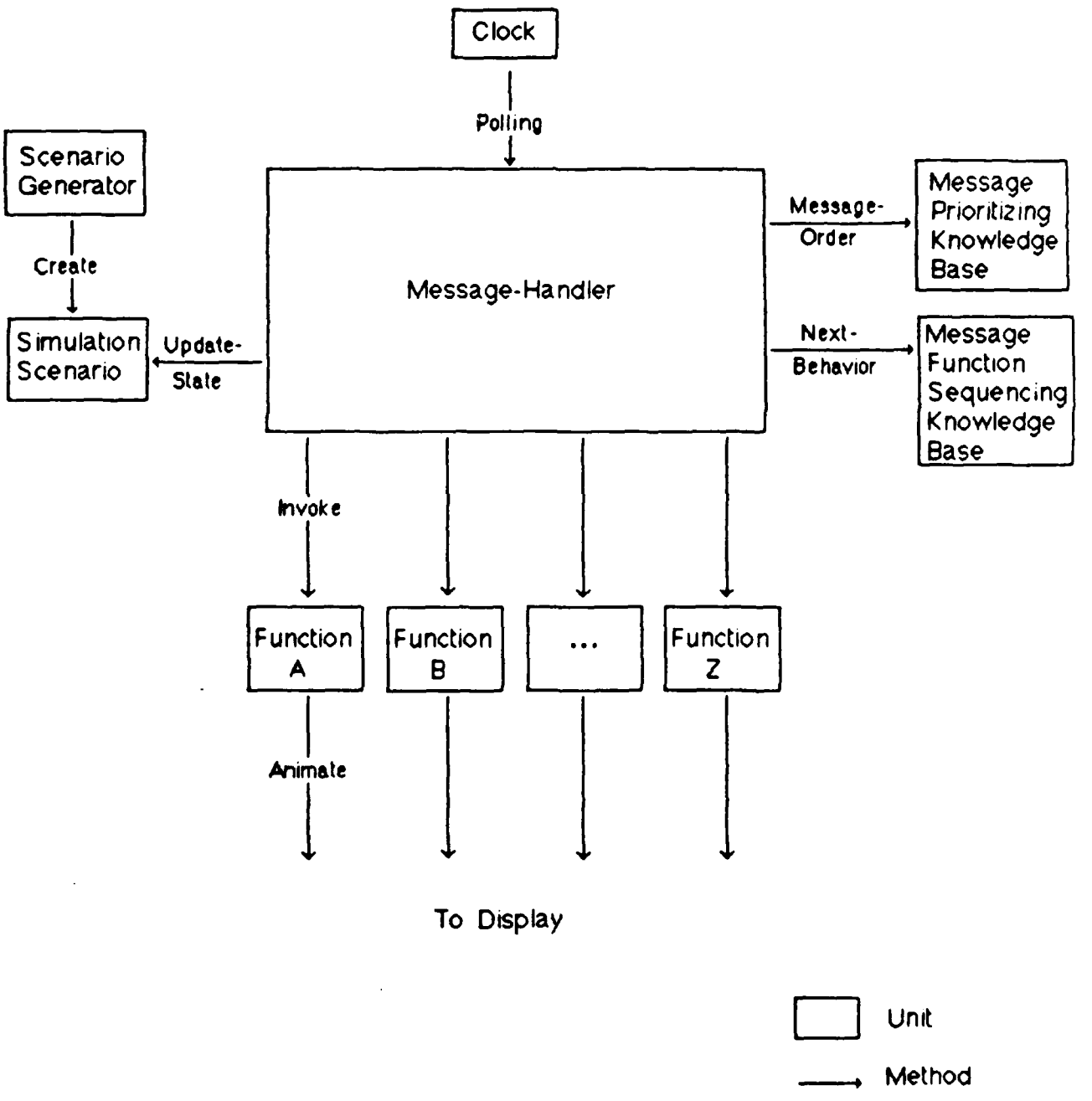

Knowledge Bases
(Output) The Graph of the WISCUS Knowledge Base
FUNCTIONS
WISCUS.FCNS
COMMODITY.SW.SUPPORT.FCNS
HOST.ACCESS.FCNS
TERMINAL.EMULATION.FCNS
MSG.PROC.FCNS
SYSTEM.ANALYZER.FCNS
SYSTEM.EXERCISER.FCNS
USER.SUPPORT.FCNS
OPERATING.SYSTEM.FCNS
SPREADSHEET.FCNS
WORD.PROCESSING.FCNS
BASIC.LANGUAGE.FCNS
DATA.BASE.FCNS
GRAPHICS.FCNS
MSG.SERVICES.FCNS
MSG.SUMMARY.QUEUE.FCNS
MSG.SUMMARY.REVIEW.FCNS
ALERT.PROCESSING.FCNS
MSG.COMPOSITION.FCNS
MSG.RETRIEVAL.FCNS
MSG.INPROCESSING.FCNS
RECEIPT.PROCESSING.FCNS
STATISTICS.UPDATE.FCN
SYSTEM.PARAMETERS.APPEND.FCN
INFO.EXTRACTION.FCN
MSG.COLLATION.FCN
MSG.JOURNALIZATION.FCN
MSG.RECEIPT.ACKNOWLEDGE.FCN
MSG.SUMMARY.CONSTRUCTION.FCN
PROFILE.MATCH.FCN
MSG.DISTRIBUTION.FCNS
MSG.REDISTRIBUTION.FCN
URGENT.DISTRIBUTION.FCN
AUTO.DISTRIBUTION.FCN
MANUAL.DISTRIBUTION.FCN
MSG.OUTPROCESSING.FCNS
MSG.RELEASE.FCNS
MSG.VERIFICATION.FCNS
MSG.COORDINATION.FCNS
WORKSTATION.WORKLOAD.SIMULATION.FCNS
CLIHOST.WORKLOAD.SIMULATION.FCNS
SYSTEM.STATISTICS.FCNS
EASE.OF.USE.FCNS
FILE.TRANSFER.FCNS
INTRA.SITE.MAIL.FCNS
SECURITY.PROCESSING.FCNS
05/02/86 13:57:33 Adler
USER:
(no window)

Figure 5

Behavioral Simulator

(Functional Requirements Description)



AI Research At Martin Marietta Baltimore Aerospace

V. Benokraitis
A. Chande
M. Dudziak
M. Hall
K. Noon

The focus of AI research in the AI and Simulations Dept. is on underwater robotics and signal processing applications. Specifically, our entire group is currently involved in the development of an Intelligent Controller for a generic class of autonomous underwater vehicles (AUV). This controller (IC) is intended for operating an AUV by receiving input from a number of on-board sensors, a world model which includes known data about ocean terrain and features and a mission plan that consists of a sequence of objectives which the vehicle must perform. The decisions made by the IC are then communicated as waypoints or destination coordinates to a real-time control system which is responsible for operating servomechanisms and end-effectors.

Figure 1 below illustrates the comprehensive design of the AUV control system, showing the place of the Intelligent Controller.

[fig. 1 goes here]

Several critical problems face the development of this IC system. First, we are dealing with classical problems in planning. More specifically, we are addressing the replanning and rescheduling of a real-time plan that will have been established by human users and provided to the AUV. This plan, composed of objectives which are in turn composed of more elementary tasks, will have a number of spatial and temporal constraints for its successful performance. In addition to the constraints imposed upon sub-plan tasks, there will be other goals and constraints which are imposed by the vehicle mechanics or the nature of environment and its missions. The job of real-time replanning becomes a job of deciphering new information about the environment, coordinating that information with what is already known about the world and the current plan, and determining what changes should be made in the original plan in order to achieve necessary system goals and mission objectives.

A second major problem is that of data fusion, correlating the inputs from a variety of acoustic sensors which will be the AUV's only interface with the actual physical environment. Limitations imposed by underwater physics and the restriction to acoustic sensing devices make this a more challenging problem than similar data fusion and object recognition tasks for terrain-following and airborne systems. Data received by individual sensors consists of range and bearing information, from which edges, surfaces and shapes must be ascertained, in order to pass information about potentially significant objects or ocean conditions to the Planner.

Martin J. Dudziak [33]
Martin Marietta Baltimore Aerospace
Mail Point 600
103 Chesapeake Park Plaza
Baltimore, MD 21220

Modelling the world is a third and very central problem domain to our project. The world modeller is initiated with a map of the known world and representations for states of the vehicle (hydrodynamics, position, subsystem diagnostics, etc.) and the ocean (thermoclines, currents). The model is updated by data which is received from the sensor acquisition system and also by data from the model which has been further interpreted by the planner.

Our approach to the problem has been to use simulation as a development tool for building and testing our intelligent controller design. The knowledge domain for the AUV problem is a sparsely populated region; our task is quite unlike building a traditional expert system since there is no strong, established body of human expertise. Simulation offers an opportunity to not only demonstrate the performance of the system but do the experimentation and exploratory programming necessary for this kind of problem.

The Intelligent Controller Development system (ICDS) is the outgrowth of this philosophy of learning by simulation. It is strongly influenced by the project development environments provided in Lisp machine workstations and in particular by our department's selected AI system-building tool, Knowledge Craft (Carnegie Group, Inc.). Through the ICDS we are designing and assembling our Intelligent Controller and testing it with modules that emulate the other functions of an AUV. Figure 2 shows how the logical parts of the ICDS fit with one another.

[fig. 2 goes here]

The ICDS is being implemented on a Symbolics 3670 with an 8-bit, 256-color graphics system. Critical software being used besides Symbolics Lisp include: Knowledge Craft and the Symbolics family of graphics packages (S-Geometry, S-Paint, S-Render, S-Dynamics). The ICDS enables developers to run a simulation which exercises the Intelligent Controller and produces a color graphics display in one of two basic modes:

3-D (showing a cubical volume of space) and
Geographic (showing an 'aerial' view with land masses and other features shown).

The displays are modifiable by a number of window-oriented menus and selectors which are accessible to the user through the main system console.

The user may, by using a menu-oriented interface, make changes to the specifications for the vehicle characteristics (e.g., hydrodynamic characteristics), world environment (e.g., placement of man-made and natural objects on the sea floor) and mission plan (objectives and specific tasks, global mission rules and goals, etc.). These changes are made using a simple text editor, enabling non-Lisp-specialists to operate the system.

Other features being built into the ICDS provide capabilities to the user for 'stop-action' viewing of internal program states and data structures and for run-time production of various history files showing the activity of the controller subsystems at different levels of detail. Figure 3 gives an illustrate of how two states of the system console will appear to the user in the course of running the ICDS simulation.

[fig. 3 goes here]

The ICDS will be operational in the later part of 1986 and is expected to play a primary role as a development tool for building the next-generation Intelligent Controller that will operate onboard a sea-going test vehicle. Our group is exploring the applicability of concurrent processor (MIMD-type) machines for meeting the real-time performance spped and interprocess communication requirements of a fieldable AUV controller.

Other areas of attention being developed as part of the MMBA long-term applied-AI thrust include the following:

- advanced signal processing and interpretation
- battle force information management
- hierarchical planning for manipulator robot control systems
- ASW multiple sensor deployment and data fusion

Our major focus is on naval and underwater applications, of which the AUV project is a crucial element. As a consequence of the systems engineering problems these projects engender, we find ourselves working on some of the most basic and long-standing issues in AI research. This is clearly challenging but also very rewarding as we take new steps forward.

--- M. Dudziak

Figure 1

5/12/86 MJD

AUV COMPREHENSIVE DESIGN

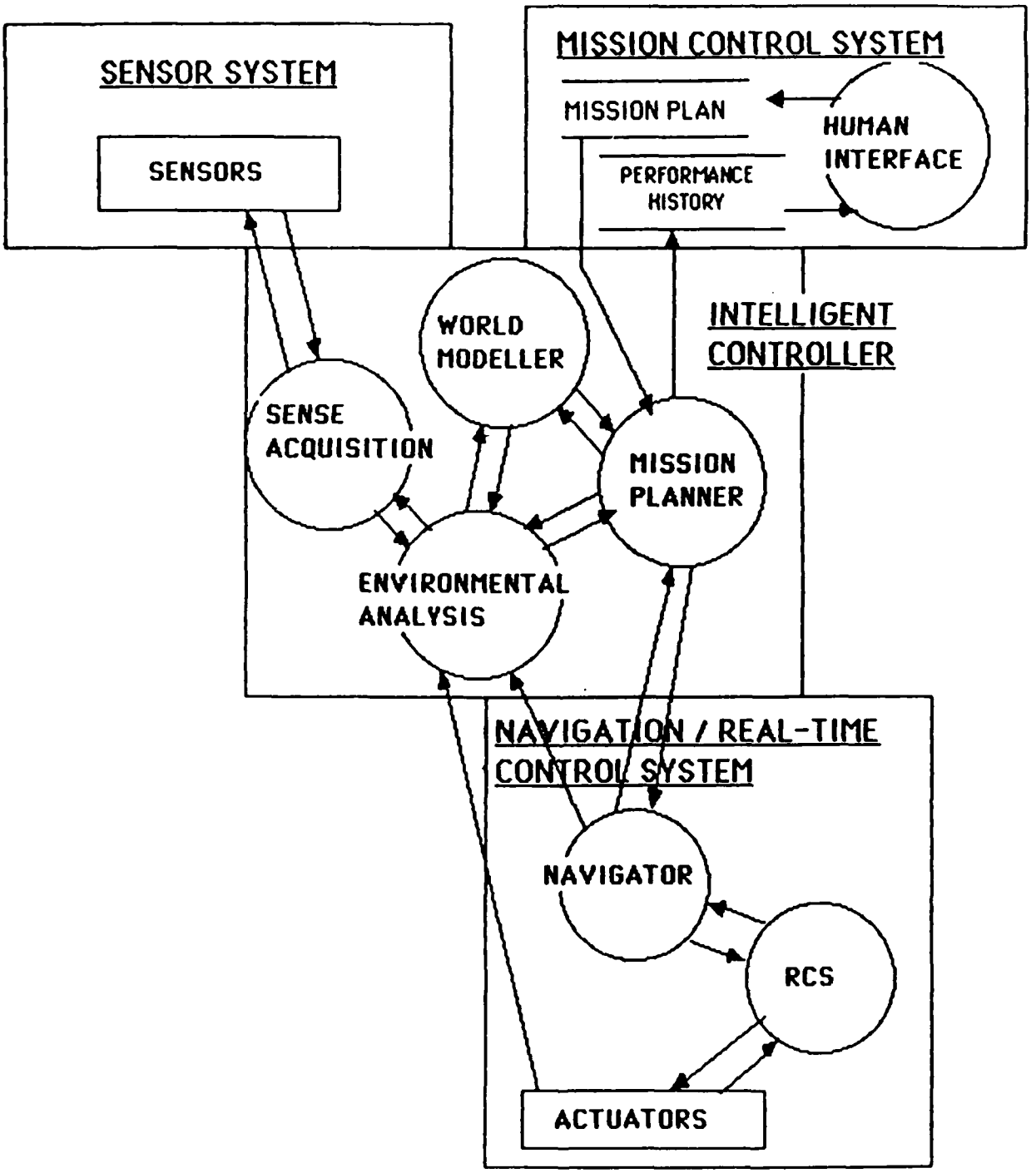


Figure 2

AUV SIMULATOR

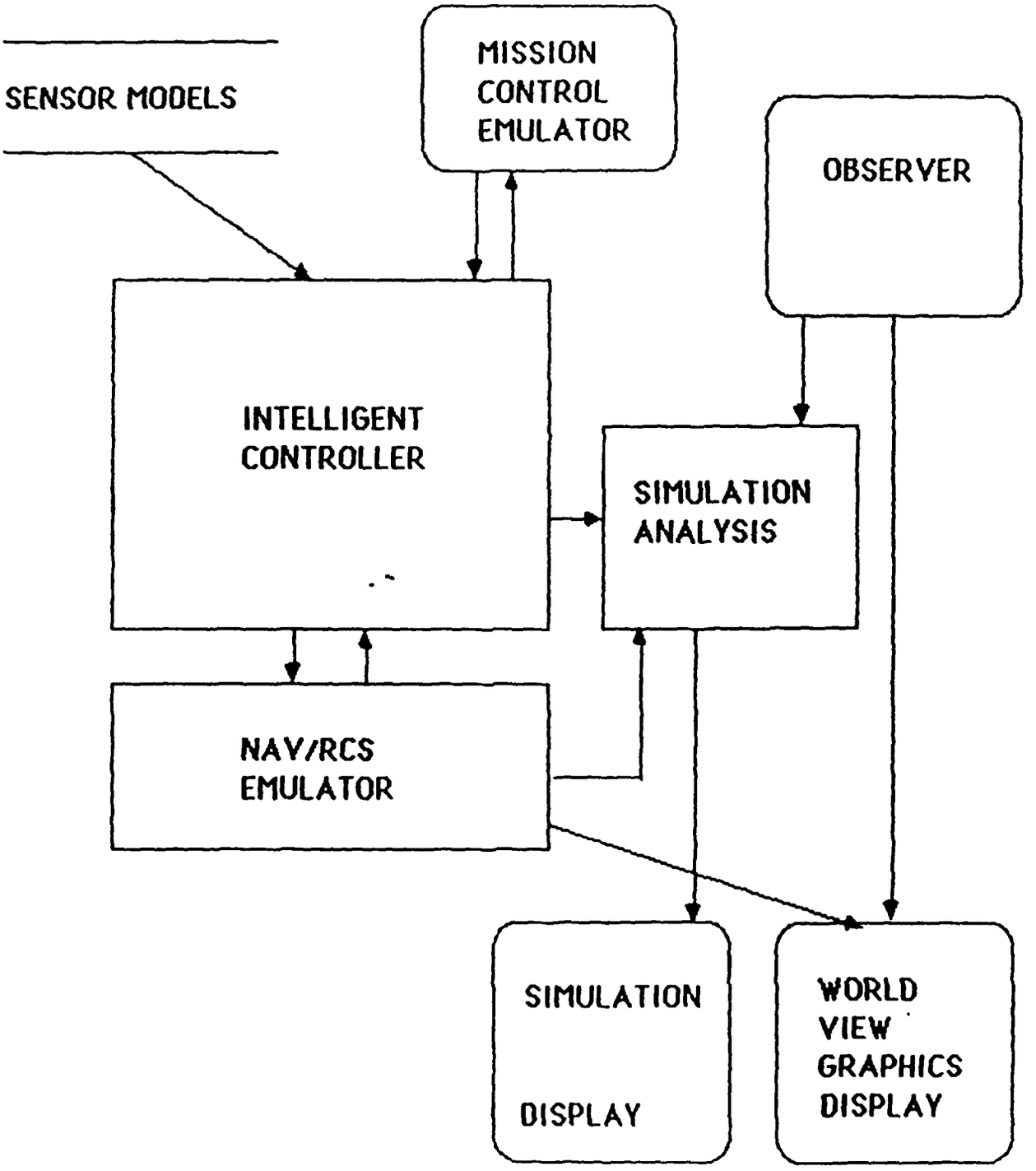
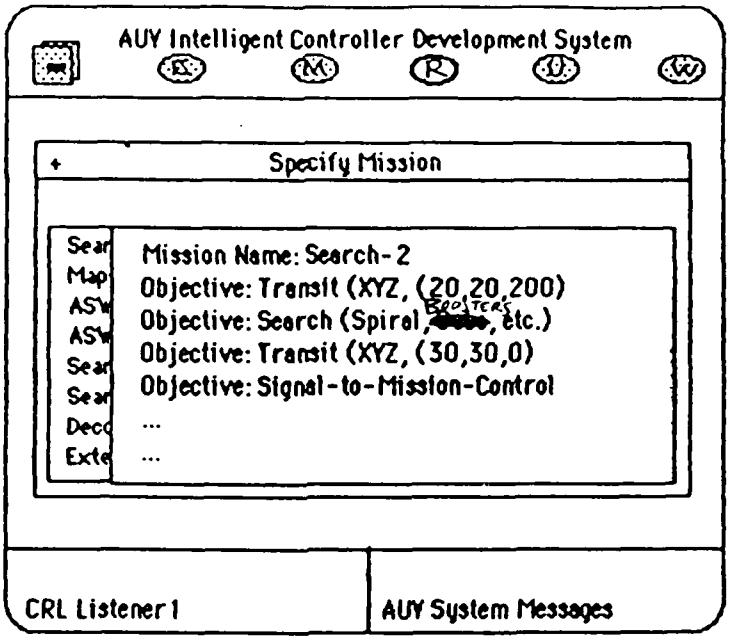


Figure 3



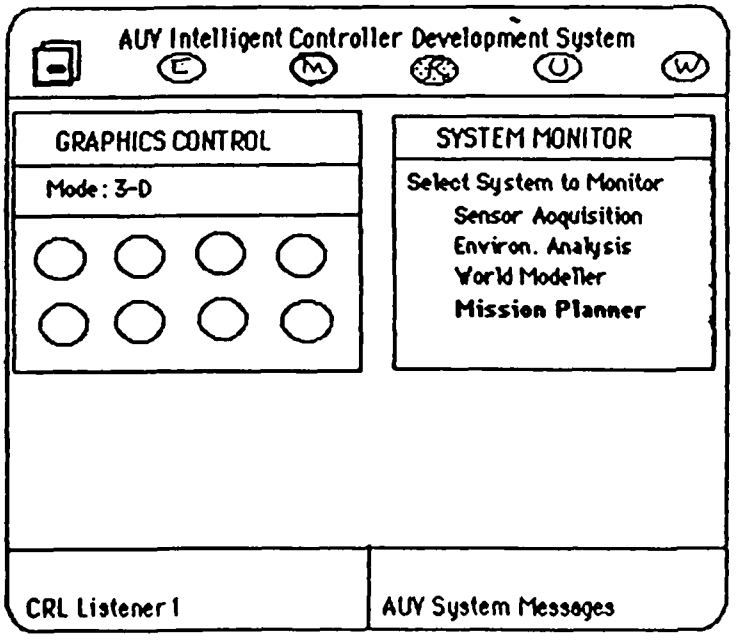
User chose to edit a new mission plan.

Note that the editor window can be enlarged incrementally by clicking on the directional arrows which are in the upper left corner of every active window for which size modification by the user is appropriate. The arrows are kept in the window (menu) at the highest level but apply to the window where the cursor is currently located.

User edits and then issues cmds to save and go up to prev. level.

Note that user can choose an existing mission and just run it as is, create a new one from scratch, or go into the Plan Editor and via a meta-x type command bring up an existing plan for editing. Similar editor commands let the user save an edited plan or abort and go back to screen 1.41.

Note also that by mouse click or keyboard code a user can always abort the current level of a task and get back to the previous level. A total abort of a task is accomplished by clicking on the current task cmd-icon.



User was in Graphics Control, moved mouse and clicked on System Monitor icon, bringing up that task's window.

Click on Mission Planner option to start monitoring activity (inter-module message traffic, entries into World Modeller, rule firings; in short, a history of the control path and data path activities). This is useful not only for development and debugging of the ICDS but also for learning what is needed for the real-world operational Intelligent Controller.

```
[Discrete-Machine
    Subclass-of
        Value:(Machine)
    Load-event
        Value:(dmload)
    Operation-event
        Value:(dmoper)
    Operation-time
        Value:(2sec)
    Unload-event
        Value:(dmunload)
    ]

[Drill
    Subclass-of
        Value:(Discrete-Machine)
    Operation-event
        Value:(drill)
    Operation-time
        Value:(10sec)
    ]
```

Figure 1: Example representation of model objects

object and all of its event methods and attributes to be stored together with the added feature of allowing a taxonomy of objects with inheritance capabilities.

A discrete-event simulator for this representation would maintain a calendar of events, each of which consists of the object of focus, its event, and the time the event should occure. The execution of an event is accomplished by sending a message to the event method at the focus object. In the frame-based representation environment discussed here the event methods can either be represented by proceedural knowledge or rule-based knowledge [5].

Truth maintenance research [4] [3] involves maintaining the truth of facts and any inferences, justifications, that the truth depends. Justifications may, in addition, be used for hypothetical assumptions as well as non-monitonic assumptions for facts. Many of these systems also maintain multiple contexts for these facts that represent multiple, possibly inconsistent, states of the knowledge base. The truth maintenance system discussed here [12] is integrated with the frame-based representation environment and incluces an extension to the traditional tree context mechanism [10]. This extension incorporates a tree context mechanism with the ability to merge contexts.

When integrated with knowledge-based simulation, this truth maintenance system provides the capabilities of backtracking, parallel simulation, and causal tracing of simulation models. The first two capabilities are made possible by using a current

context, where all actions of the model take place, and by checkpointing model execution at various intervals. Checkpointing is accomplished by spawning a new context which is a child of the current context. Execution of the model is then continued within this new current context. An example of this is seen in figure 2, where each context,

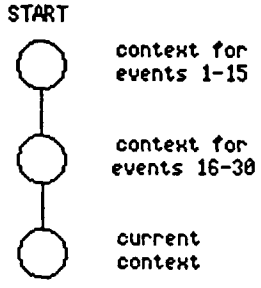


Figure 2: Checkpointing

world, contains all of the events and their subsequent actions for 15 consecutive events. At the end of 15 events, checkpionting takes place, a new context is spawned and made the new current context. Of course, the interval of 15 events was arbitrary.

Backtracking is accomplished by interrupting the model, spawning a new context off of some previous context, and making it the new current context. Execution of the simulation model can continue from the new point. Changing the current context in this situation essentially changes the state of the model to some previously saved state. An example of this can be seen in figure 3. This mechanism can be used, for example, to spawn new contexts from some saved state and make structural and/or parametric changes. With these changes made, the model can be run and then compared to the results of models without the changes. Also the results of model changes can be seen without having to completely rerun the model from the beginning.

Parallel simulation is accomplished by maintaining multiple states of the model in separate contexts. The execution of the model can be swapped between the contexts intermittently. That is, the simulator executes in one context, advancing the state of the model in that context, for some predefined interval of time or number of events,

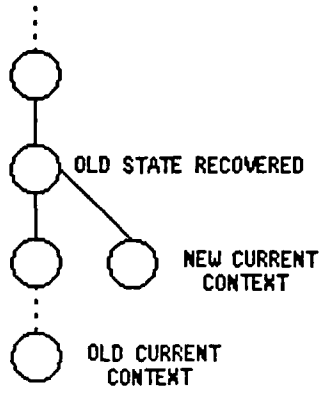


Figure 3: Backtracking

then swaps contexts and executes in another context for the same interval. This concurrent execution of multiple models allows for real-time comparative analysis of simulation models.

Causal tracing of the simulation model is accomplished through the recording of justifications. Actions taken within an event can be justified by the event itself as well as any conditions on that action. Also scheduled events can be justified by the action scheduling the event. By recording these justifications a complete causal tracing of any aspect of the model execution can be accomplished. These causal tracings can then be graphed as seen in figure 4.

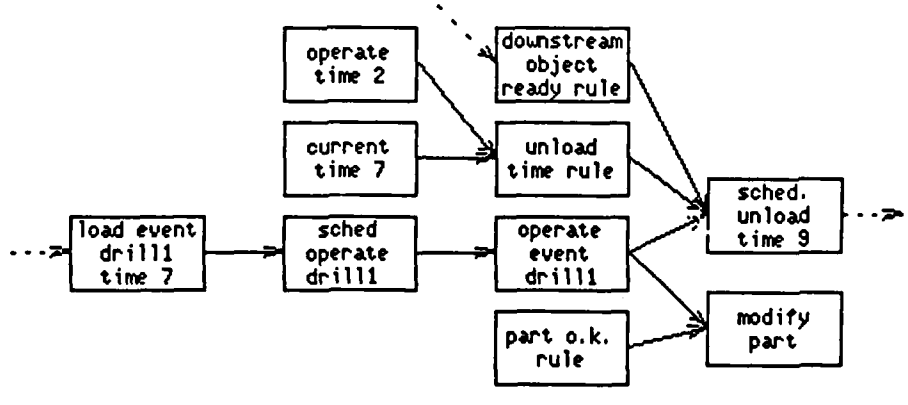


Figure 4: Causal trace

Causal tracing could be used for model verification, model learning by the user, and

possibly behavioral abstraction. One approach to causal tracing for knowledge-based simulation [1] was to collect, within an event, all data references and modifications and causally link them to that event. A major drawback is the black-box assumption of causation between an event and the references made within the event. To deturmine a better estimate of causation, this data was collected over many event executions and statistically analyzed. The current approach avoids that inaccuracy and inefficiency by using a truth maintenance system which can record each action and all of the inferences leading to those actions within any event.

References

[1] Baskaran, V. & Reddy, Y.
An Introspective Environment for Knowledge-Based Simulation.
In *Proc. Winter Simulation Conf.*. Austin, Texas, 1984.

[2] Dahl, O.-J.
SIMULA: An algol-based simulation language.
Communications of the ACM 9:671-678, 1966.

[3] de Kleer, J.
An Assumption-Based Truth Maintenance System.
Artificial Intelligence 28(2), 1986.

[4] Doyle, J.
A Truth Maintenance System.
Artificial Intelligence 12(3), 1979.

[5] Fikes, R. & Kehler, T.
The Role of Frame-Based Representation in Reasoning.
Communications of the ACM , September, 1985.

[6] Goldberg, A., & Robson, D.
Smalltalk-80: The language and its implementation.
Addison-Wesley, Reading, Massachusetts, 1983.

[7] Klahr, P. & Faught, W.
Knowledge-Based Simulation.
In *AAAI*. 1980.

[8] Kunz, J., Kehler, T., Williams M.
Applications Development Using A Hybrid AI Development System.
AI Magazine 5(3), 1984.

[9] McBride, P. and Gross, G.
SIMKIT: A Knowledge Based Discrete Simulation System.
In *CAD-CON*. 1984.
Presentation.

[10] McDermott, D. & Sussman, G.
The Conniver Reference Manual.
Technical Report A.I. Memo No. 259, MIT AI Laboratory, May, 1972.

[11] Minsky, M.
A Framework for Representing Knowledge.
The Psychology of Computer Vision.
McGraw-Hill, New York, 1975.

[12] Morris, P. and Nado, R.
Representing Actions with an Assumption-Based Truth Maintenance System.
In *Proceedings of the AAAI*. 1986.

[13] Reddy, Y.V. & Fox, M.S.
KBS: An Artificial Intelligence Approach to Flexible Simulation.
Technical Report CMU-RI-TR-82-1, The Robotics Institute, Carnegie-Mellon University, 1982.

AN AUTONOMOUS VEHICLE WORKSTATION

Dan Donahue

Artificial Intelligence Laboratory
Computer Science Center
Texas Instruments, P.O.Box 226015
MS/238
Dallas, TX 75266
(214) 995-0668

Over the past three years, TI has been developing an Autonomous Vehicle Workstation (AVW). The purpose of the AVW is to provide a framework and simulation environment for testing algorithms to control autonomous vehicles. Currently, the system is being used to support a robotic air vehicle program, a pilot's associate program, and a multisensor fusion project. The AVW consists of a simulated world, a platform, and various "crew members" that communicate with the simulated world through the platform. The current set of crew members consists of a premission planner, an observer, a pilot/commander, and a platform controller. The simulation environment coordinates the updating of the crew members, the platform, and the simulated world.

The simulated real world consists of a surface representation (either a terrain map or mathematical model), individual object models, weather definitions, and various methods for interfacing the platform with the world. Objects in the world are built from a hierarchy of basic object definitions. The objects currently available include various air and land vehicles, stationary objects for representing landmarks, transmitters, etc., and line and area objects such as roads, fields, and ridges. This library of object definitions can be used by developers to create their own specialized objects to be used in a world environment. As an example, a user interested in developing a model of a highway can use the predefined definition of a 'road, and add attributes and methods to this definition which are unique to highways.

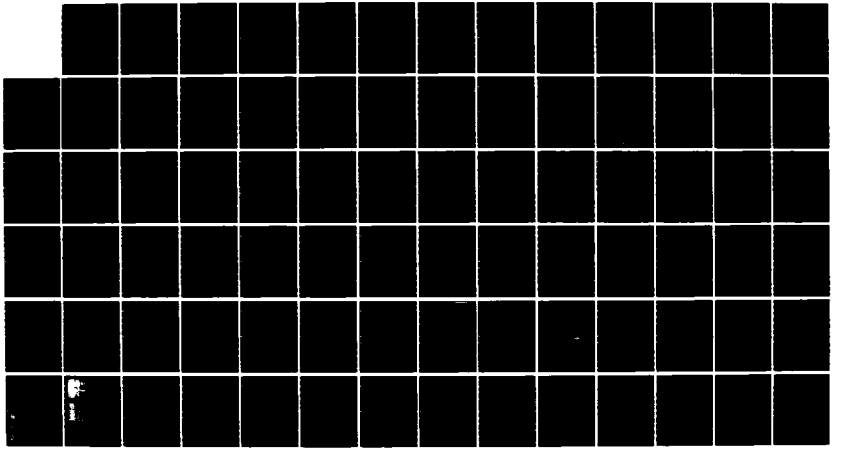
A platform model is constructed from a number of subsystem models, such as a flight control system, comm-nav system, stores system, and sensor system. Basic models for each of these subsystems are provided and may be expanded depending on the requirements of a particular mission to be simulated. The AVW currently provides some detailed subsystem definitions for air vehicles that behave like an F-16.

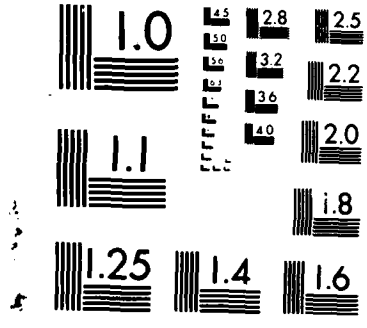
The premission planner creates "safe" paths from one point in the simulated world to another. Safe paths are determined by consideration of several aspects of the premission state of the world, such as, terrain, user supplied mission goals, and a priori knowledge of obstacles. The AVW provides templates for the other crew members that provide the minimum functionality needed to run simulations. Users of the AVW develop and test AI algorithms within the simulation environment to increase the functionality of the crew members.

A configuration and graphics manager controls the display of information to the user. An optional color screen can be used to display additional information. The displays can be configured as desired to demonstrate the functionality of individual crew members or the state

AD-A174 053 AAAI (AMERICAN ASSOCIATION ON ARTIFICIAL INTELLIGENCE) 2/2
WORKSHOP ON AI (AR..(U) ARMY CONCEPTS ANALYSIS AGENCY
BETHESDA MD A GERSTENFELD ET AL. AUG 86

UNCLASSIFIED F/G 9/2 NL





of the platform, simulated world, and simulation environment as a whole. A dashboard is provided to change and display the state of the platform. Various methods of viewing the simulated world that are provided include an "out of the canopy" view, an "eye in the sky" view, and a plan/contour view.

The AVW executive manager regulates the flow of information and communication between processes running in the simulation environment. The simulation environment allows for processing modules (i.e., crew members, platform, simulated world) to reside and run on one or more processors. The environment is general enough to run a simulation consisting of an arbitrary number of processes distributed over an arbitrary number of Lisp machines.

Developed on TI Explorer Lisp Machines, the AVW is almost completely object oriented. All modules and sub-modules are instances of particular object definitions, providing applications programmers with easy and efficient means to quickly develop systems which either replace or augment the basic default crew member modules which exist in the AVW. Since the objects all communicate by means of message passing, module interfaces can be be designed which allow easy insertion of user's applications into the overall AVW system.

Among the specific areas currently being addressed in the development of the AVW are:

World Object Definition: The problem of representation of data in the simulated world environment is being investigated. Methods of storing or retrieving positional data in terms of latitude and longitude in addition to Cartesian coordinates are being developed. Issues concerning display and retrieval of "world information" are being investigated, including digital map display techniques. Also, we are interested in world modeling techniques...i.e. given the simulated "real world", how do we best represent a reasonably accurate model of that world?

Inter-module communications: We are addressing in depth the problems of distributed processing in multi-machine environments. In particular, we are developing mechanisms which will allow developers to have "crew member" modules residing on separate machines and communicate with the simulated world environment and other crew member modules via inter-machine message passing.

Time: The representation of the flow of real time is a central issue in the development of the simulation capabilities of the AVW. With the possibility of different system modules residing on separate machines, the need to coordinate their time with all other modules' becomes paramount. We are developing a scheduling module which insures that each module has the same notion of time; i.e. one module does not think that 10 seconds have elapsed while all other modules think that 20 seconds have elapsed.

As mentioned above, the AVW is currently being used as a simulation testbed for a number of projects, including a robotic air vehicle project (RAV). The RAV group has developed their own jet aircraft vehicle object built up from the basic default object definitions provided by the AVW. Using this specialized vehicle definition, the RAV group is developing an expert piloting module which will attempt to fly the vehilce in the AVW's simulated world. This piloting module and a "pilot-vehicle interface" reside on two machines while the AVW resides on a third machine. Using information concerning the state of the aircraft (obtained

from the vehicle in the AVW environment), the piloting module makes high-level flight decisions (i.e. bank left, climb 45, etc.). These "flight plans" are sent to the pilot-vehicle interface which in turns sends low-level commands (i.e. stick forward 2, left rudder 4 etc.) to the vehicle object in the AVW environment.

Simulation-Based Robot Cell Diagnosis

C. N. Lee, M. Y. Chiu, S. C. Clark, P. Y. Liu

Artificial Intelligence Department
Research and Technology Laboratories
Siemens Corporate Research & Support, Inc.
105 College Rd. East
Princeton N.J. 08540

1. Introduction

Design and diagnosis of robotic assembly cells are important tasks for factory automation, especially for automatic assembly line design. Robotic assembly cell failures may down a factory production line and decrease its productivity. An appropriate diagnosis system, whether it is an on-line or off-line diagnosis system, is required [Barbera84] [Chiu85] to maintain high productivity by reducing the production line down-time caused by a robot cell failure. This research explores *simulation-based robot cell diagnosis.*

Problem characterization is carried out with a laundry pump motor assembly cell. The functional structure and operations of the robotic assembly cell is qualitatively simulated with the model of *synchronous parallel operation.* A *causal network* is constructed to represent the causal relations between the robot control program and the physical cell environment. A *graphic process editor* and *an automatic information translator* are system design tools. *Causal reasoning* is mainly used and heuristic rule-based reasoning is used for conflict resolution in deadlock cases.

2. Robot Cell Modeling and the Cell Failure

Robot cell operations are characterized as synchronized parallel operations. The robot and the physical cell environment, i.e., various parts carriers, fixtures etc., operate in mutually synchronized manner. Figure 1 shows two synchronized processes.

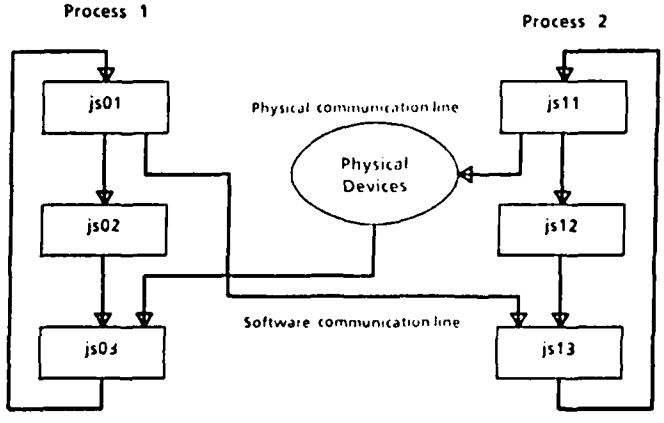


Fig. 1 Synchronous parallel operation example with two communication lines.

Each process consists of a circular sequence of tasks. Once the preconditions of a task are satisfied, the process can start and execute that task. The task may be finished without any

interruption or may be suspended to wait for other conditions to be satisfied during the execution period. Each process is synchronized to related processes. Synchronization between processes is obtained by sending messages through communication lines.

In the robotic assembly cell, various actuators an sensors play the role of communication line. For example: the control program outputs a control signal, this control signal is amplified to actuate a solenoid, which in turn actuates an air valve. By actuating the air valve, the related mechanical action occurs and is noticed by a sensor. The output of the sensor is routed back into the control program. Thus, the robot cell can be represented as a causal network linked by the process task sequence and by these communication lines. A robot cell failure occurs when one or more of the communication lines break. In such a case, processes get into a deadlock state waiting for signals from each other. Therefore, robot cell diagnosis, in this context, is first identifying the broken communication line and then locating the faulty physical device of that line.

However, there are two questions to be answered in this approach for the diagnosis system design. The first is how to get the diagnosis system model; the system model can be constructed in the robot cell design stage or can be constructed based on the flowchart of the robot control program and the existing cell environment. In the design stage, the assembly flowchart or state transition diagram can be directly mapped into a synchronous parallel operation model. However, the latter is considered to generalize the diagnosis system design. The second question is how much detail each communication line must be modeled; this cannot be answered clearly yet, it is a question of required diagnosis level. The more detail the diagnosis gets into, the more detail of each physical device model is needed. This can be expanded to the electronic equipment diagnosis level for each device or it can be simply modeled as an object variable of the control program. The electronic equipment level diagnosis is not yet fully considered.

3. System Implementation and Reasoning

Two kinds of information are utilized for the system implementation. One is the robot control program and the other is the information about the physical cell environment. The system model is constructed by performing process definition and process segmention for task sequence generation on the robot control program. Then the necessary information for robot cell diagnosis is automatically extracted from the robot control program and incorporated into the system model. From the cell environment all the causal relationships of physical devices are obtained. The causal relationships of physical devices in the system model can be checked with a simulation module.

Once the system model is obtained, the system reasoning is straightforward in the simulation-based causal reasoning [Davis84] [Dekleer83]; it starts reasoning in data-driven fashion. All the available current status information is collected in the beginning of the diagnosis. Suspected communication lines are selected using the current status data, i.e., process status, software flags, and sensor information. If no more data is available for diagnosis, then heuristic rule-based reasoning is applied on the local device. The present system reasoner has a hierarchy, which corresponds to system model structure. Top level reasoner localizes the failure in the process level and passes the result to the local reasoner. Then the local reasoner locates the broken communication line and the corresponding physical device. Device-specific reasoners are also being considered in local reasoning.

4. Conclusions

Fifteen faulty cases were generated by a robot cell engineer; physical sensors were cut off or assembly parts were removed intentionally. All cases were diagnosed correctly in the level of task and communication line.

However, propagateds fault are not diagnosed completely. When one of the physical switches was deactivated, the diagnosis system localized the fault in the physical device block. Since the sensor and the activator are modeled as one physical device block in the present system, the system could not detect the exact faulty device. Also, if a sensor is blamed then it

should be checked whether the sensor is physically broken or if the sensor activation did not occur. The system simulation model also could not resolve this ambiguity. It can be diagnosed correctly by human operator intervention or by running the real robot cell with the proper device testing program. No built-in tests or physical tests are considered in the present system.

Although the system is in a refinement stage and the number of case analyses is not enough, it is shown that the simulation-based causal reasoning can localize the fault to the physical device level and the system can be expanded easily to include various physical device diagnosis. Causal reasoning provides a means of propagated error analysis. Also, system design aids such as a graphic process editor and automatic information translator proved to be necessary for information intensive diagnosis system design.

Acknowledgment

The authors would like to thank all members of the Robotics group who shared their opinions, especially to A. Marcantonio who provided valuable discussions about robot cell structure and operations.

References

[Barbera84]
A. J. Barbera, M. L. Fitzgerald, J. S. Albus and L. S. Haynes, "RCS: The NBS Real-Time Control System", ROBOTS 8 Conference Proceedings, Detroit, Michigan, 19.1-19.62, June 4th, 1984

[Chiu85]
M. Y. Chiu and E. Niedermayr, " Knowledge-Based Diagnosis of Manufacturing Cells" Siemens Research and Development Reports, Vol. 14, No. 5, 230-237, 1985

[Davis84]
Randal Davis, "Diagnosis via Causal Reasoning: Paths of Interaction and the Locality Principle", AAAI-84 Proceedings, 88-94, 1984

[Davis84]
Randal Davis, "Diagnostic Reasoning Based on Structure and Behavior", in: Qualitative Reasoning about Physical Systems, The MIT Press, Cambridge, Massachusetts, 347-410, 1984

[DeKleer83]
Johan de Kleer and John S. Brown, "Assumptions and ambiguities in mechanics mental models", in: Mental Models, Erlbaum, Hillsdale, New Jersey, 155-190, 1983

Device Modeling and Simulation in an Automatic Programming System

Sol J. Greenspan

Schlumberger-Doll Research
Old Quarry Road
Ridgefield, CT 06877-4108

203-431-5000

submitted to

Workshop on AI and Simulation
August 11, 1986

Device Modeling and Simulation in an Automatic Programming System

Sol J. Greenspan

Schlumberger-Doll Research
Ridgefield, CT 06877-4108

Introduction

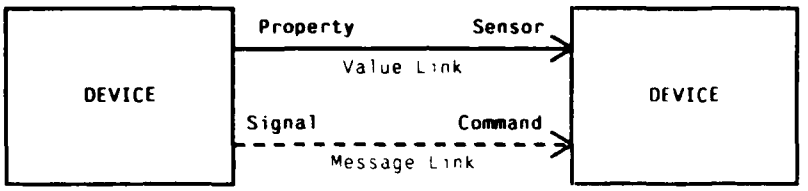
My interest in simulation pertains to my research in the area of applying AI techniques to Software Engineering. A basic premise underlying this work is that a system that automates significant parts of the software development process must be domain-specific in order to interact effectively with a user who is an expert in the application domain but computationally naive. [Barstow85] Thus, facilities are needed for representing and reasoning about domain knowledge. My current work investigates these problems in the context of software that controls remote devices (e.g. oil well logging tools). For such software, a key part of the needed domain knowledge is knowledge of the device itself, for which we have developed a knowledge representation formalism. An important requirement on the device representation is that it support simulation.

There are two reasons why simulation must be supported. The first is for software engineering purposes: to facilitate analysis of software requirements, and to form part of the software testing environment. The second is to facilitate the reasoning processes involved in writing the software. Below I will briefly describe the knowledge representation formalism and then discuss these two uses.

The Knowledge Representation Formalism

The world consists of a set of interacting devices. Each device may be considered as both a primitive and as a composite object. As a primitive, a device's behavior is described in terms of states and transitions. As a composite, a device is described by a set of interconnected, component devices. Every device has well-defined interfaces that dictate how it may be connected to other devices.

The method for representing primitive devices is a finite-state formalism augmented by an assertion language. A device has a set of attributes, each a function of continuous time and of other attributes. A state node has a predicate over attributes of the device, which is supposed to be true of a device whenever the device is in that state; since the predicate may involve time, it can specify not only an invariant on the state but also some behavior that is to take place in that state. A transition node has a pair of predicates on the attributes; when the first (the trigger condition) becomes true, the transition fires after which the second (the result condition) is asserted.



In a composite device description, there are four kinds of interfaces connected by two kinds of links. A property interface of a device makes the value of one of the device's attributes continuously available to any connected value links. A sensor interface of a device makes the value of one of the device's attributes constantly equal to the value coming in on the value link. A signal interface of a device is a place where messages, the values of one of the device's attributes, are emitted at discrete moments in time when some assertion becomes true. A command interface of a device is a place where messages are received and made the value of one of the device's attributes at discrete moments in time, resulting in an assertion's becoming true at that moment. A link has an associated time delay, so that the value of a property at time t is available at the sensor at time t plus a delay, and a message sent at time t is received at time t plus a delay.

The following are some points that we think are interesting about this model:

1. Clean interfaces separating devices. Device description is solely in terms of its private attributes. Links have no semantics other than passage of time.

2. Continuous links used for accurate physical description. Simulation can be done to any desired precision and is not predetermined by the description. The same is true for continuous time: simulation can be executed to any desired level of time granularity

3. Equivalence of within-device and between-device semantics. A message link is like a transition, described by two assertions, but with a time delay. A value link between devices says that an

attribute of one device is a function of an attribute of the other, the value of the latter being time-delayed.

Simulation for requirements analysis and testing

Using AI knowledge representation techniques for describing requirements has been discussed in [Greenspan84] and [Borgida85]. In the case of device control software, it is obviously hard to think about requirements for controlling a device without understanding how the device works. The ability to simulate the behavior of the device significantly enhances one's ability to understand the device and also to understand what the device control software should make it do.

As a starting point for an experimental implementation, we have devised a graphical editor for representing devices in the formalism described above. Placing graphical objects on the screen causes the creation of objects in a device knowledge base, including all the structure needed for the simulation. The graphical objects are also the basis for animating the simulation: manipulations of the device representation cause user-programmed changes on the screen.

Each device or component of a device has its own local time attribute ("clock"). Advancing a composite device to some future time t involves advancing each of its components until the furthest-behind device has advanced to t. Advancing a primitive device at some resolution involves doing the actions implied by the assertions associated with states, detecting trigger conditions, and firing transitions. We still need to consider what kinds of simulation control, e.g. breaking, undoing, and backtracking are needed, based on what information we want to glean from the simulation.

Simulation, of course, can also be used as a basis for testing the device control software. It is often much easier to test software with a simulated device than in the real environment (e.g. with oil well logging instruments in a borehole). [Barstow86]

Reasoning About Devices

We are interested in automating parts of the software development process that we call formalization, which begins with an informal specification of what the device control software must do and results in a formal specification. An informal specification is an imprecise, incomplete description of the program to be written, while the formal specification is a precise description of

what the program should do but not how it should do it. (Such concerns as computational tractibility and efficiency are relegated to a different part of the software development process, implementation, which involves going from a formal specification to code in some target language) The device model gives a formal model in terms of which to express the formal specification. Moreover, we believe that there is sufficient information in our model to enable a large part of the process to be carried out automatically. For example, a requirement might be stated as getting the device from an initial state into some desired state. First, the state-transition diagram can be analyzed to find possible paths. This gives a sequence of conditions that must be made true of the attributes in order to accomplish the goal. The problem is to then figure out what commands of the device can cause these conditions, and finally, what signals can be emitted by the controller to cause these commands to be executed.

References

[Barstow85]
Barstow, D. R., "Domain-Specific Automatic Programming", *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 11, November 1985, pp.1321-1336.

[Barstow86]
Barstow, D. R., Barth, P., and Dinitz,R., "Exploiting the Differences Between Programming Environments and Runtime Environments", (to appear).

[Borgida85]
Borgida, A., Greenspan, S. J., and Mylopoulos, J "Knowledge Representation as the Basis for Requirements Specifications", IEEE Computer, April 1985, pp. 82-91.

[Greenspan84]
Greenspan, S. J., "Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition," (Ph.D. Thesis) CSRG Report 155, Dept. of Computer Science, University of Toronto, 1984.

[Greenspan86]
Greenspan, S. J. and Barstow, D. R., "Using a Device Model as Domain Knowledge in the Automatic Programming of Software to Control Remote Devices" (available from authors).

THE AI-SIMULATION GROUP
DEPT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF ARIZONA
Bernard P. Zeigler
Jerzy R. Rozenblit

Our research activities concentrate in the following directions:

1. Unified Simulation Knowledge Representation

The world views of discrete event simulation are highly compatible with the representation schemes of Artificial Intelligence (O'Keefe, 1986). Object-oriented programming (LOOPS (Bobrow & Stefik, 1983), FLAVORS (Weinreb et. al, 1983), TI Scheme (1986)) can be viewed as providing a computational basis for the frame hierarchy (Minsky, 1975) by allowing the programmer to associate methods with objects which are inherited just as other slots are. It is not surprising therefore, that languages are being developed to express both the dynamic knowledge of discrete event formalisms and the declarative knowledge of AI frame paradigms (Klahr, 1986; Reddy et. al. 1986).

Our research objective has been to unify these developments in a more fundamental paradigm which draws its inspiration from the systems theory view of the world (reviewed by Pichler, 1985) and stems from the system theoretic representation of simulation models for multifacetted modelling methodology (Zeigler, 1986).

System theory distinguishes between system structure (the inner constitution of a system) and behavior (its outer manifestation). Regarding structure, the theory has given us the concept of decomposition, i.e., how a system may be broken down into component systems, and coupling, i.e., how these components may be combined to reconstitute the original system. Thus decomposition and coupling should be fundamental relations in a knowledge representation scheme. System theory however has not focussed itself on a third fundamental kind of relation, taxonomic, which concerns the admissible variants of a component and their specializations, exhibitted for example, by the generalization hierarchy of frames.

As a step toward a complete knowledge representation scheme, we have combined the decomposition, taxonomic, and coupling relationships in a representation scheme called the system entity structure (Zeigler, 1984), a declarative scheme related to frame-theoretic and object-based representations.The model base contains models which are procedural in character, expressed in classical and AI-derived formalisms mentioned earlier. The entities of the entity structure refer to conceptual components of reality for which models may reside in the model base. Also associated with entities are slots for attribute knowledge representation. An entity may have several aspects, each denoting a decomposition and therefore having several entities. An entity may also have several specializations, each representing a classification of the possible variants of the entity. The generative capabilty of the entity structure enables convenient generation and representation of model attributes at multiple levels of aggregation and abstraction.

A set of tools has been developed to facilitate development and operations on system entity structures.

2. Use of the System Entity Structure in Simulation Model Synthesis

A primary application of the above knowledge representation scheme is to objective-driven development of simulation models (Zeigler, 1984). In this approach, a simulation model is synthesized from components in the model base based on modelling objectives. The objectives guide a pruning process which reduces the entity structure to one or more composition trees from which models may be hierarchically built up from atomic components. Constraints placed on the aspects of the entity structure restrict the family of possible pruned structures for more informed search. The approach requires that the model base be populated by models in the DEVS formalism, a modular discrete event model representation enabling hierarchical assembly (Zeigler, 1984). An environment of this kind is being developped for the domain of local area network simulation, intended to enable a designer to rapidly assess the performance of a wide variety of alternative network designs via model synthesis and simulation (Sevinc, 1986).

3. Application to Distributed Simulation

Research in distributed simulation, the use of multiprocessor architectures as simulation media, has concentrated thus far on execution of a single model on a network of processors. Yet in practice, simulation experimentation most often involves exploration of the behaviors of a related family of models. Our research aims to develop a methodology, centered on the entity structure/model base concepts, to map a variant family of models onto a hierarchical simulator. Several Ph. D. theses have developped the ground work for this approach supported by NSF grants (Concepcion, 1984; Baik, 1985; Rozenblit, 1985). In current work, we intend to extend this methodology to enable variable structure simulations, in which the model structure may be changed according to intelligence embedded within the model itself. A demonstration project of this kind has been completed in which a work-load adaptive multiprocessor architecture is simulated in SIMSCRIPT, a conventional discrete event language (Zeigler, 1986). A prototype of a distributed hierarchical simulator, in which variable structure models may be specified, has been completed in the SCHEME language, using its object oriented and "first class" object handling features.

References

Adelsberger, H.H. et. al.(1986), "Rule Based Object Oriented Simulation Systems", In: Intelligent Simulation Environments, (eds: P.A. Luker & H.H. Adelsberger), Simulation Series, Vol. 17, SCS, San Diego, CA.

Baik, D. (1985) Performance Modelling and Optimal Model Decomposition in Distributed Simulation, Ph. D. Dissertation, Wayne State University, Detroit, MI.

Belogus, D. (1985) Multifacetted Modelling and Simulation: A Software Engineering Implementation, Doctoral Dissertation, Weizmann Institute of Science, Rehovot, Israel.

Bobrow, D.G. (1976), "Dimensions of Representation", In: Representation and Understanding: Studies in Cognitive Science, (eds: D.G. Bobrow & A. Collins), Academic Press, NY.

Bobrow, D.G. (1985), Qualitative Reasoning about Physical Systems, MIT Press, Cambridge, MA.

Bobrow, D.G. & M.J. Stefik (1983), The LOOPS Manual, Xerox Corporation, Palo Alto, CA.
Franta, W.R. (1977), A Process View of Simulation, North Holland Pub. Co.

Concepcion, A. (1984), Distributed Simulation on Multiprocessors: Specification, Design, and Architecture, Ph. D. Dissertation, Wayn State University, Detroit, MI.

Futo, I (1985) "Combined Discrete/Continuous Modeling and Problem Solveing", AI, Graphics and Simulation, (ed: G. Birtwistle), SCS Pub., San Diego, CA.

Hogeweg, P. & B. Hesper (1986), "Knowledge Seeking in Variable Structure Models", In: Modelling and Simulation in the Artificial Intelligence Era, (eds: M.S. Elzas, T.I. Oren, B.P. Zeigler), North Holland Pub. Co., Amsterdam.

Intellicorp (1984), "The Knowledge Engineering Environment", Menlo Park, CA.

Kerckhoffs, E.J.H. & G.C.Vansteenkiste (1986), "The Impact of Advanced Information Processing on Simulation- an Illustrative Review", Simulation, 46:1,17-26.

Klahr, P. (1986), "Expressibility in ROSS, an Object-Oriented Simulation System", In: Artificial Intelligence in Simulation (eds: G. C. Vansteenkiste, E.J.H. Kerckhoffs & B.P. Zeigler), SCS Publications, San Diego, CA.

Klir (1985), Architecture of Systems Problem Solving, Plenum Press, New York, 539 p.

McDermott, J.(1982) "R1: A Rule-based Configurer of Computer Systems", Artificial Intelligence, 19.

Minsky, M. (1977), "Frame-System Theory", In: Thinking (eds: Johnston-Laird & Wason), Cambridge U. Press, Cambridge, 355-376.

O'Keefe R. (1986), "Simulation and Expert Systems- A Taxonomy and Some Examples, Simulation, 46:1, 10-16.

Oren, T.I. (1986a), "Knowledge Bases for an Advanced Simulation Environment", In: Intelligent Simulation Environments, (eds: P.A. Luker & H.H. Adelsberger), Simulation Series, Vol. 17, SCS, San Diego, CA.

Oren, T.I. (1986b), "Implications of Machine Learning in Simulation", In: Modelling and Simulation Methodology in the Artificial Intelligence Era", (eds: M.S. Elzas, T.I. Oren & B.P. Zeigler), North-Holland Pub. Co., Amsterdam.

Pichler, F. (1985) "Dynamical Systems Concepts", in: Source-book in Cybernetics and Systems Research (ed: R. Trappl) , hemisphere Pub. Co.

Rajagopalan, R. (1986), "The Role of Qualitative Reasoning in Simulation", In: Artificial Intelligence in Simulation (eds: G. C. Vansteenkiste, E.J.H. Kerckhoffs & B.P. Zeigler), SCS Publications, San Diego, CA.

Reddy, Y.V., M.S. Fox & N. Husain (1985), "Automating the Analysis of Simulations in KBS", Proc. SCS Multiconference, San Diego, CA.

Reddy, Y.V., M.S. Fox, N. Husain, & M. McRoberts (1986), "The Knowledge-Based Simulation System",IEEE Software, March, pp 26-37.

Robertson, P. (1986), "A Rule Based Expert Simulation Environment", In: Intelligent Simulation Environments, (eds: P.A. Luker & H.H. Adelsberger), Simulation Series, Vol. 17, SCS, San Diego, CA.

Rozenblit, J.W. (1985),A Conceptual Basis for Model-Based System Design, Doctoral Dissertation, Wayne State University, Detroit, MI.

Rozenblit, J.W. & Zeigler, B.P. (1985), "Concepts for Knowledge-Based System Design Environments", Proc. Winter Simulation Conference, San Francisco, CA.

Standridge, C.R.(1986) "Simulation Data Bases", in: Encyclopedia of Systems and Controls, (ed: M. Singh), Pergammon Press (to appear),

Struik, D.J. (1967), A Concise History of Mathematics, Dover Pub., New York.

Texas Instruments Inc. (1986), TI Scheme Language Reference Manual, Dallas, TX.

Weinreb, D., D. Moon, & R. Stallman (1983), Lisp Machine Manual, MIT, Cambridge, MA.

Winograd, T. (1976), "Frame Representations and the Declarative/Procedural Controversy", In: Representation and Understanding: Studies in Cognitive Science, (eds: D.G. Bobrow & A. Collins), Academic Press, NY.

Zeigler, B.P. (1985), "System-theoretic Representation of Simulation Models", IIE Transactions,16:1, 19-34.

Zeigler, B.P. (1984), Multifacetted Modelling and Discrete Event Simulation. Academic Press, London.

Zeigler, B.P. (1986), "Toward a Simulation Methodology for Variable Structure Modelling", In: Modelling and Simulation Methodology in the Artificial Intelligence Era (eds: M.S. Elzas, T.I. Oren, B.P. Zeigler), North Holland Pub. Co., Amsterdam.

Zeigler, B.P. & R. Reynolds (1985a), "A Hierarchical Information Processing Model for Adaptation to Technology Change", [illegible]

Zeigler, B.P. & R. Reynolds (1985b), "Towards a Theory of Adaptive Computer Architectures", Proc. 5th Int. Conf. Distributed Comp. Sys., Denver, CO.

Zeigler, B.P., D. Belogus & A. Bolshoi (1980), "ESP- An Interactive Tool for System Structuring", Proc. Eur. Mtg. Cby. Sys. Res., Hemisphere Press, USA.

Zeigler, B.P. (to appear), "Knowledge Representation from Minsky to Newton and Beyond", Applied Artificial Intelligence, Vol 1.

Current Research in Qualitative Simulation

Peter D. Karp

Knowledge Systems Laboratory

Computer Science Department

Stanford University

Stanford, CA 94305

For the AAAI-86 Workshop on Simulation

Introduction

The goal of the Molgen group at Stanford is to build a computer system which can reproduce the discoveries of Prof. Charles Yanofsky of the Stanford Biology Department. Prof. Yanofsky discovered a novel mechanism of gene regulation in bacteria. For a program to make these discoveries, it must be able to represent an initial theory of this domain, plus new theories it generates given the experimental results which are its inputs. These theories describe how bacteria regulate their expression of a certain group of genes based on the concentration of nutrients in the cell's environment.

Our group has begun to apply the techniques of Qualitative Simulation (QS), as explored by [De Kleer 84, Iwasaki 85, Forbus 84, Davis 84], to the problem of expressing these theories. Applying these techniques to a real-world problem has led us to identify limitations of these techniques in addition to those which have previously been identified [Kuipers 85]. Our current research attempts to rectify some of these shortcomings. It has led us to re-consider QS from a more general perspective, which suggests natural means of augumenting its techniques. We begin by describing this perspective.

A Perspective on Qualitative Simulation

A number of motivations have been expressed for the work on QS, most of which address shortcomings in classical quantitative simulation techniques, such as:

1. Quantitative information about a particular system or problem instance is often not available, yet people are often able to make sensible predictions without it.

2. Quantitative simulations often require large amounts of computation time.

3. The belief that quantitative information often obscures the "essence" of a situation and cannot be used to generate explanations of device behavior.

4. The desire to develop a naive physics of the world to comprise part of the common sense knowledge of intelligent systems.

We share all of these motivations save the last one. The theories which we will represent should exhibit not naive problem solving performance, but expert-level performance.

We will shortly consider each of these motivations in further detail, but we first describe the core ideas of QS.

In quantitative simulations one identifies the set of parameters or state-variables for the system one wishes to model, writes a set of equations which describe the interactions between these parameters, determines initial values for certain key parameters, and cranks them through the equations to derive values for the other parameters. QS uses exactly the same approach, with two exceptions. First, instead of giving parameters quantitative values, QS variables have qualitative values such as {-1,0,1} or {low,normal,high}. A qualitative

calculus is then defined to determine how these values are propagated through constraint equations. The second difference is that classical quantitative simulation programs have a procedural representation (e.g., Fortran code), while QS programs have a declarative representation which can be reasoned over.

Current Goals of our Research

We now consider how the above techniques relate to the described motivations, point out shortcomings, and indicate how our work will address them.

A more general statement of goal (1) is that programs should be able to solve problems using varying amounts and types of information, producing the best solution possible given the information available. They thus must be able to represent all the information which might pertain to a given problem. Current QS programs reason with one type of information: qualitative values. [Kuipers 85] indicates the limits of this approach; our domain in particular requires the increased precision which Kuipers shows QS cannot provide, and some quantitative information is available. (Forbus' Quantity Space representation [Forbus 84]) provides a partial ordering over qualitative values, and in is a step in the right direction, but more expressiveness is required still.)

We are thus exploring the space between quantitative and qualitative information for other techniques which can represent the information available, for example, a number times a qualitative value, e.g., "what happens if I double the normal concentration of this enzyme?" Note that our viewpoint implies that a simulator should be able to accept a mix of values with different types of precision, i.e., some values might be quantitative, some qualitative, some in between.

Next, consider that just as information about parameter *values* might be imprecise, so the *relations between* parameters might not be known precisely. Current approaches write arithmetic constraints between parameters, e.g., **P = Q1A1 + Q2A2** (Pressure equals a sum of flows times areas). In fact, knowledge about the form of a relation varies; we might know:

- Only that **A** varies as **B**, i.e., when **B** increases, so does **A**.
- **A** varies *linearly* as **B**.
- **A** varies linearly as **B** with a *known constant*.

Our simulation system (based on KEE) views both parameters and the relations between them as complex entities which are represented using one or more frames. The information stored about a relation indicates whether it is bi-directional or uni-directional, its sign, the mathematical form of the relation if known, and the values of any relevant constants, if known. If the mathematical form of the relation is not known, corresponding values of its parameters under landmark experimental conditions can be recorded, along with rules for

interpolating between these values. Biologists appear to use this technique when reasoning about enzyme kinetics problems, for example.

Parameters can have one or more types of values (qualitative, quantitative, or the combination discussed above). We also represent knowledge of their landmark values for certain experimental conditions, their normal ranges of values, and their units of measurement (e.g., "molecules per cell"). These provide constraints on the values the parameters can take, and the relations they enter into (i.e., units must combine correctly). They also make it easier for us to change our simulation clock.

Currently we are working on code to actually propagate these parameters through these expressive relations to allow us to predict how the quantities of different molecules changes in the cell over time under different experimental conditions.

Techniques which have been developed to achieve goal (2) include limit analysis [Forbus 84] and aggregation [Weld 85] - both involve induction over a cyclic simulation to predict future changes without simulating many intermediate states. We plan on building a simulation which can be run at several levels of detail *independently* to provide both faster problem solving and to help make the discovery problem more tractable [Karp 85, Karp 86]. Some work has been done in this area [Davis 84], but there is still a large space of possible simulator designs to be explored [Karp 86]. For example, objects, or the processes acting on them, or some combination of the two, could be represented at multiple levels of detail; what is the appropriate choice? How does one choose the appropriate level of detail for solving a given problem? Can a description at one level of detail be compiled or expanded automatically to produce a description at another level of detail?

We have two hypotheses regarding goal (3). First is that the explainable nature of QS systems is probably not due to their use of qualitative values, but to their use of declarative representations: declarative quantitative simulations could also be explained (though perhaps not as simply). Second is that a qualitative simulation will almost always imply naive performance; we believe that most human experts utilize more precise information in addition to their qualitative models. Our research will help us evaluate these hypotheses.

Summary

Our approach to QS can be summarized by saying that while a number of techniques for QS have been developed, much work remains to be done in exploring how they fare when confronted by real-world problems - or how to confront real-world problems with a coordinated combination of these techniques. In addition, by producing a generalized description of these techniques we will gain a better understanding of how they relate to one-another and to quantitative simulation.

References

[Davis 84] Davis, R.
Diagnositic reasoning based on struture and behavior.
Artificial Intelligence 24(1-3):347-410, December, 1984.

[De Kleer 84] De Kleer, J. and Brown, J. S.
A qualitative physics based on confluences.
Artificial Intelligence 24(1-3):7-84, December, 1984.

[Forbus 84] Forbus, Kenneth.
Qualitative Process Theory.
Technical Report AI-TR-789, MIT, 1984.

[Iwasaki 85] Iwasaki, Y. and Simon, H. A.
Causality in Device Behavior.
Technical Report, CMU, 1985.

[Karp 85] Karp, P.
Thesis Proposal: Qualitative Simulation and Discovery in Molecular Biology.
Technical Report KSL-85-36, Stanford University, 1985.

[Karp 86] Karp, P., and Wilkins, D.C.
An Analysis of the Deep/Shallow Distinction Between Expert Systems.
Technical Report KSL-86-32, Stanford University, 1986.

[Kuipers 85] Kuipers, B.
The Limits of Qualitative Simulation.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 128-136. August, 1985.

[Weld 85] Weld, D.S.
Combining Discrete and Continuous Process Models.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 140-143. August, 1985.

Workshop on AI and Simulation

Kasim Sinnamohideen
Controls Research
Johnson Controls, Inc.
Milwaukee, Wisconsin

Stephen Zvolner
Controls Research
Johnson Controls, Inc.
Milwaukee, Wisconsin

The principle goal of our research activity into qualitative simulation is to represent a significant subset of knowledge that experienced building operators use in their daily activities in operating and managing the heating, ventilating, and air conditioning (HVAC) systems of a building. These mechanical systems include chillers for supplying cooling, boilers for heating, and fan systems for ventilation, among others.

Operators running physical plant equipment make use of qualitative models in helping them to understand and properly operate the plant equipment. The rationale for our investigation into qualitative simulation methods is to attempt to emulate the reasoning process that an experienced building operator uses in these tasks.

The qualitative models developed will serve a multiplicity of purposes including support for diagnostics, building operations support, operator training, intelligent explanation, enhanced information presentation, and ultimately, machine aided learning.

This work will draw heavily upon the significant contributions in qualitative reasoning by researchers like Hayes, Johan de Kleer, Ken Forbus, Chuck Reiger and Milt Grinberg, and others. Our work shares the common goal of attempting to formalize the commonsense knowledge about the every day physical world. Specifically, within the context of our application domain, our goal is to formalize that subset of commonsense physical knowledge dealing with typical engineered systems used in controlling the building's environment.

What distinguishes our approach to qualitative simulation is that we seek to encode, explicitly, a broad base of pragmatic engineering knowledge particular to the HVAC domain. Thus, we place less emphasis on the microscopic details of the behavior of physical systems, and instead, emphasize the macroscopic aspects of behavior. We have coined the term Qualitative Engineering to distinquish our approach to qualitative reasoning.

Qualitative Engineering borrows some of the epistemological ideas of Reiger and Grinberg; the notion of states, enablements, tendencies, etc. Unlike the work of Reiger and Grinberg, though, our work does depend on the integration of function and structure as an integral part of the representation. Furthermore, we have extended the notion of connectivity to include both physical as well as functional connections. Thus one can talk about electrical connections, fluid connections, thermodynamic connections, etc.

Another distinction between our approach to qualitative reasoning is that we do *not* rely on the use of qualitative differential equations to express behavior. Rather, it is our goal to develop a concise, but powerful, representation of HVAC controls knowledge using a small set of heuristics to describe qualitatively the physical behavior of building HVAC equipment and systems. In the Qualitative Engineering representation, for example, the fact that closing the valve will stop the flow is represented explicitly as a rule instead of a qualitative differential equation.

The goal of this project is to develop and demonstrate a methodology for constructing qualitative simulations of HVAC equipment and systems. Our current activities address the representation of the physical components themselves (e.g., pumps, valves, pipes, heat exchangers); the representation of collections of components into subsystems and systems; the representation of the behaviors of these components in a machine executable form (e.g., the purpose of a pump is to supply pressure); the representation of the structural and functional connections between components; and developing constraint propagation mechanisms to "compute" the system's behavior based upon this knowledge.

Steven A. Epstein
Management Analysis Co.
12671 High Bluff Dr.
San Diego, CA 92130
26 MAY 1986

RiTSE: The Reactor Trip Simulation Environment

Management Analysis Company, located in San Diego, is a general consulting corporation dealing almost exclusively with nuclear power generation.

During the spring of 1985, while investigating the causes of inadvertant actuation of reactor trip signals at the Salem Nuclear Plant, we developed RiTSE, the Reactor Trip Simulation environment. RiTSE is a frame and rule based AI system which represents all components, processes, and their interactions and current states. An event driven inference engine predicts if a contemplated action, such as taking a pump out for servicing, would cause a reator trip, or any other event or system state that had been identified as 'interesting'.

One of the principal causes of 'unplanned' outages and poor emergency response is human error resulting from the subtle and complex interactions of the many simultaneous activites of maintenance, testing, and surveillance at the plant. These activities often affect systems or components at a very deep level. Indeed, maintenance and operations personnel can change the operating state of nearly every component in the plant.

Typically, operations and maintenance personnel are not trained to analyze the potential interactions and ramifications

of their activities. Those responsible for authorizing such activities are often quite knowledgeable and may understand the plant to the extent of hundreds of sub-systems and components. However, it is quite impossible for any individual to remain aware of the tens of thousands of components, their potential interactions, and the impact of the next activity on the system as a whole.

The effect of subtle, or complex, interactions among nuclear plant components has historically been difficult to analyze. Indeed, in response to recent incidents at home and abroad, the U.S. NRC appears to be reemphasizing treatment of system interactions, particularly with respect to the effect of so-called non-saftey syatems on safety systems.

Therefore our engineers wanted a system that could (1) allow the plant staff to predict if an action or set of actions would cause a reactor trip, and (2) should a trip occur, aid in determination of the components and systems which directly contributed to the trip.

Traditional solution methods, stemming from reliability technology, require that one knows beforehand which components interact and what the ramifications might be. However, the very problem is to discover those interactions, not to assume them.

Using RiTSE, we model each component individually, looking

one step downstream, as it were. For example, the rule for a motor operated valve might be:

```
MOV1 is OPEN if
    BREAKER1 is CLOSED &
    480VACBUS is ENERGIZED
            or
    BREAKER2 is CLOSED &
    480VACEMERGENCY is ENERGIZED

       OTHERWISE

MOV1 is CLOSED
```

In this manner we consider all aspects of a component: its power supplies, its control mechanisms, its associated instrumentation, and both the effect of dynamic processes on the component and the component's impact on processes.

We employed a frame structure representation for both plant entities and rules. We populated our model of the nuclear plant with frame types such as mechanical component, electrical component, controller, process, sensor, system, and functions (which are rules which refer to no plant entity).

We have used a hybrid approach to rule representation. A rule is much more than a simple 'if-then' pair. There are many slots in a rule frame, only one of which has the 'rule'; other slots include English of the rule, pointers to referants, names of heuristics, actions to take, values to return, and conflict resolution instructions.

RiTSE essentially tinker-toys the plant together. As a delighted user remarked, it is a boolean spreadsheet.

Our system was used at the Salem Nuclear Plant with great ease by the engineers involved, and produces accurate results. We have spent the past year demonstrating our system and delivering conference talks in the nuclear field all over the world. Several requests for systems have resulted, including the Atomic Energy Council of the Republic of China's request for a system for the Maanshan Unit 1.

We have found that we have developed a generalized framework within which to model nuclear power generataing facilities. In the obvious hindsight of Chernobyl, the need for an interactive tool to assist nuclear plant operators in assesing the impact of proposed actions becomes clear. Our tool is a first step in that direction.

We discovered some practical issues in the development of AI systems: "twinference", when a rule for a component's state must also be executed with an eye towards operability, and the fortuitous use of databases for instantiation of frames.

We are presently investigating the relationship between deep representations and shallow inference, and in turn shallow representations and deep inference: the logon versus the analogon (λογον vs. αναλογον.

Proposal to attend the AAAI Workshop on Qualitative Process Simulation, August 1986.

QUALITATIVE SIMULATION OF CHEMICAL PROCESS SYSTEMS

Olayiwola O. Oyeleye and Mark A. Kramer

Department of Chemical Engineering

Massachusetts Institute of Technology

Cambridge, MA 02139.

Telephone: (617) 253-6508

PROJECT OBJECTIVES

In a modern chemical process plant, the operators supervisory control activities include the implementation of start-up, shut-down and change-over procedures, process monitoring and malfunction diagnosis. The operator must understand the behavior of the plant in order to accurately predict the consequences of control actions and select the correct responses.

The supervisory control decisions made by process operators are frequently based on training, but often they are based on mental models of the plants behavior. These models are usually developed by experience and are not rigorous and consequently may be inaccurate and incomplete. Mental simulation of hypothetical situations may involve long and complex chains of qualitative reasoning and can place significant cognitive load on the operator during time-constrained situations. This provides the motivation for developing computer simulation tools for assisting the operator

in making supervisory control decisions.

Traditional algorithmic computer programs based on detailed mathematical models yield the most accurate results but require large amounts of process data and long computational times. Furthermore, quantitative predictions can be undesirable as insightful concepts are qualitative in nature and the quantitative predictions may be difficult to interpret. Human experts are able to make accurate qualitative predictions of the consequences of their control actions and equipment malfunction. Therefore, the thrust of this research is toward qualitative modeling of process behavior.

In the chemical engineering domain, the underlying phenomena of momentum, heat and mass transfer, kinetics, process chemistry and thermodynamics are well understood. A model-based system is required to capture this knowledge. The goal of our research is to identify the principles that underlie qualitative process behavior and to begin to understand and to create computer simulations of the process of qualitative process simulation. The simulator developed could serve as the model-based "core " of expert systems addressing various supervisory control tasks.

RESEARCH PROGRESS

Our initial attempt to model the qualitative behavior of processes has been to investigate qualitative malfunction simulation. The effect of a fault on a continuous steady-state process is to cause a deviation from its normal steady-state condition. Dynamic effects have been ignored and the ultimate directional

changes in the state of the process after fault or disturbance initiation are predicted by propagating the effects of the fault through a qualitative model of the process. The application of these predictions is in diagnosis of faults, using a fault/symptom table, or a hypothesis/test diagnosis strategy. The paper describing these results is attached.

The view of qualitative process behavior advanced in our work is that the effects of faults are causally propagated from one variable to another while satisfying process constraints. Therefore, methods and models used for fault simulation should include both concepts of causality and constraint satisfaction.

Causality is represented in our method by the signed directed graph (SDG) (Umeda et al. 1980). The SDG consists of a set of nodes and signed arrows process, representing the immediate cause and effect relationships among process variables. Process constraints are introduced through qualitative equations (confluences) (De Kleer and Brown, 1984). Confluences are logical constraints representing multiple opposing tendencies among process variables and not causality.

Qualitative simulation is a fundamentally underspecified problem and potentially leads to multiple solutions. Some of the solutions obtained may be spurious and others may be genuine, and valid for a system with a certain set of numerical parameters. The basic challenge in all qualitative simulation methods is to reduce the ambiguities generated by the removal of quantitative information.

A novel feature of our work, the ambiguities are reduced

by the addition of "latent" constraints associated with mass, energy and information feedback loops in the process. These constraints are derived from redundant quantitative process equations. While it cannot be proven that any technique will eliminate all spurious solutions (Kuipers 1985), our technique has not produced any in all the examples we have tried. Our method does not require numerical parameter values and has been tested on small scale realistic processes involving interacting control loops, mass and energy feedback, and multiple conflicting causal pathways between proccess variables. In cases where multiple solutions were predicted, they were valid for certain realizations of system parameters.

Our current method has some limitations. Dynamic information is ignored and in processes that exhibit inverse response, directional changes in the values of some variables during the transient may be opposite to that predicted at steady state. It also ignores the possibility of entering regimes where physical behavior undergoes fundamental change and the SDG and the confluences no longer apply. Further work needs to be done to resolve these complex issues.

REFERENCES

De Kleer, J. and J.S. Brown (1984). A qualitative physics based on confluences. Artificial Intelligence., 24, 7-83.

Kuipers, B. (1985). The limits of qualitative simulation. Proc. 9th IJCAI, 128-136.

Umeda, T., T. Kuriyama, E. O'Shima, and H. Matsuyama (1980). A graphical approach to cause and effect analysis of chemical processing systems. Chem. Eng. Sci., 35, 2379-2388.

53

PETRI NET SIMULATION OF RULE BASED SYSTEMS

by

Allen Rude
[illegible]

[illegible]

Allen Rude [53]
LMSC
295 Sequoia Road
Boulder Creek, CA 95006

1.0 INTRODUCTION

For twenty years Petri Nets have provided an analysis and modeling tool for the theoretical study of systems. The use of Petri Nets in industry, however, has been limited because they are difficult to implement algorithmically.

The rich theory associated with Petri Nets yields analysis techniques to determine safeness, boundedness, conservation, liveness, reachability, and coverability. These techniques have resulted in theoretical applications in the modeling of hardware and software processing, machine-shop scheduling, management scheduling, and even chemical transformations. Petri Nets seem especially well suited for the modeling of conflict and concurrency. Real world applications, however, have never proven as successful as the theoretical applications.

Efficient commercial environments for building rule based systems now allow a natural use of Petri Nets and their corresponding models. Additionally, the convention of object orientation aides the implementation of Petri Net models.

2.0 OBJECT ORIENTATION

Object orientation is an attempt to parallel real world objects with software objects. Real world objects are physical or conceptual. The translation of the conceptual objects used by experts defines the development of an expert system. Current commercial expert system products provide frames to capture data objects and methods/flavors to capture functions bound to data. The objects of

basic transitional logic, however, must be captured in rules. These rules can be represented by Petri Nets.

The objects of transitional logic consist of pairs of states: local pre-states, and local post-states. In fault diagnosis, for example, a local pre-state would be an observed anomally in a machine component. A local post-state would signal the need for component tests. The nodes of a Petri Net correspond to local states. Transitions link the states, defining the logical causality between them. In this manner, the Petri Net provides a visual and theoretical representation of the conceptual states and state transitions in an expert's solution to a problem.

3.0 TRANSLATION TO A RULE BASED SYSTEM

Experts tend to break problems into components. The nodes of a Petri Net model those components. Experts study problem components to determine their logical relationships. Petri Net transitions model those relations.

The implementation of Petri Models in algorithmic software is difficult because standard algorithms tend to capture a transient states. They are often contrived and hard to understand. Typical implementations rely heavily on table representations which either require much storage or limit the size of the nets. In either case, the speed of simulation decreases rapidly with the size of the network.

Object orientation overcomes these difficulties by forming a direct correspondence between nodes, transitions, and software. The software grows with the network. Hardware and software tailored to

object orientation and symbolic processing (such as the Flavor System) allow efficient simulation by localizing object processes. For example, statistics may be kept locally (by the software objects which represent a node or a transition) eliminating the interface between local objects and calculation modules.

In algorithmic implementations, graphic representations of Petri Nets are separated from the net definition and so require an interface. Object oriented representations on Symbolic processors include graphical representations with the net definitions, allowing sophisticated MMI in which the user sees the actual implementation.

Rules are an object oriented representation of an expert's logical analysis of a problem. The left hand side (hypothesis) of a rule represents the local state of the problem before the rule fires. The right hand side of a rule (the conclusion) represents the local state after the rule fires. The rule itself captures the logical relation of the local states. In this manner, rule based systems are often used to model an expert's solution to a problem in the same manner as Petri Net models. A rule based interpretation of Petri Nets is natural: nodes correspond to the left and right hand sides of rules while transitons correspond to the rules themselves. Using rule based systems to implement Petri Net models applies all the power of symbolic processing developments to Petri Net simulation, while maintaining the logical causality of real world solutions. Petri Nets provide a visual link between the expert analysis and software rule implementation, allowing both the developer and expert to build a system incrementally and view or simulate that system as it is developed.

The Dining Philsophers problem (suggested by Dijkstra in 1968) exemplifies the relation between a Petri Net representation and a rule based system. Five pholosophers are seated in a circle with a single chopstick between each one. The philosophers alternately think and eat. In order to eat, however, a philosopher must use two chopsticks: he must take one from each of his neighbors.

The Petri Net representation of this problem is shown in figure one. Different states and solutions to the problem can be modled by placing tokens in differrent positions.

The rule based representation can be defined directly from the Petri Net in which active rules correspond to active transitions. This rule based system consists of five pairs of rules (one for each philosopher). Each pair of rules consists of a philosopher-eat rule and a philosopher-think rule. The left hand side of the philosopher-eat rule requires three facts: the philosopher is thinking, the philosopher has a chopstick on his left, and the philosopher has a chopstick on his right. The right hand side results in three actions: the removal of both chopsticks, the philsopher begins to eat. In a similar manner, the second rule changes an eating philosopher to a thinking one (figure two).

Just as the placing of tokens can simulate various situations in this Petri Net model, the assertion of facts in a rule based model can simulate various alternatives.

4.0 ABSTRACTION

An important feature of (extended) Petri Net models is the ability to model at various levels of abstraction. Any section of a

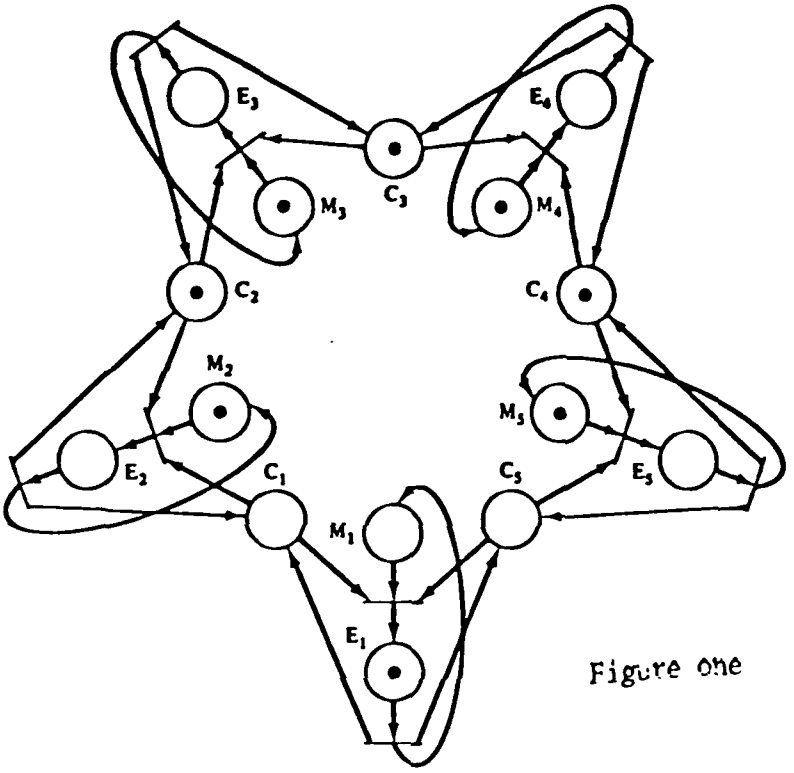


Figure one

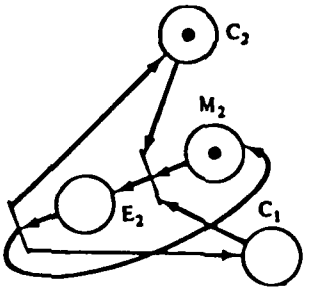


Figure four

DINING PHILOSOPHER RULES

```
(RULE Philosopher-1-Eat
          ?f1 <- (Philosopher-1-State Thinking)
          ?f2 <- (Chopstick-1 Available)
          ?f3 <- (Chopstick-2 Available)
=>
          (Retract ?f1)
          (Retract ?f2)
          (Retract ?f3)
          (Assert (Philosopher-1-State Eating)))
```

Eating Rule for Philosopher-1

```
(RULE Philosopher-1-Think
          ?f1 <- (Philosopher-1-State Eating)
=>
          (Retract ?f1)
          (Assert (Philosopher-1-State Thinking))
          (Assert (Chopstick-1 Available))
          (Assert (Chopstick-2 Available)))
```

Thinking Rule for Philosopher-1

Figure two

Petri Net between two nodes may be modeld by a single transition with an algorithmic delay. This allows well understood parts of a problem to be modeled quickly and inobtrusively.

A corresponding second level of abstraction in a rule based system uses *frames instead of specific facts*. In the Dining Philosopher problem, for example, the five sets of rules are replaced by one set. The left side of the philosopher-eat rule is replaced by a frame whose slots correspond to the philosopher's name, his state (thinking or eating), his left chopstick (whether or not it is in use), his right chopstick, his left neighbor, and his right neighbor (figure three). The right side changes not only the values in the current philosophers's frame, but those of his neighbors (he takes their chopsticks). The corresponding Petri Net is show in figure four. Just as a more abstract Petri Net must algorithmically model both the network and tokens abstracted, a more abstract rule set must model the facts (slots in a specic frame) and the fact relations (slots in different frames).

5.0 CONCLUSION

The natural correspondence between Petri Nets and rule based systems suggests that Petri Nets may be used as analysis and management tools for rule based systems.

The logic of rule based systems is often hard to follow because they are not algorithmic or sequential. The graphical representation of a Petri Net aides the understanding of the logical relations of a corresponding rule based system. The representation may be at various levels of abstraction.

DINING PHILOSOPHER RULES

```
(RULE  Philosopher-eat
            (SCHEMA ?Philosopher-Name
                        (STATE Thinking)
                        (LEFT-CHOPSTICK  Available)
                        (RIGHT-CHOPSTICK Available)
                        (LEFT-NEIGHBOR  ?Philosopher-Left)
                        (RIGHT-NEIGHBOR  ?Philosopher-Right))
   =>
   (MODIFY (SCHEMA ?Philosopher-Name
                        (STATE Eating)
                        (LEFT-CHOPSTICK Busy)
                        (RIGHT-CHOPSTICK  Busy)))
   (MODIFY (SCHEMA ?Philosopher-Left
                        (RIGHT-CHOPSTICK Busy)))
   (MODIFY (SCHEMA ?Philosopher-Right
                        (LEFT-CHOPSTICK Busy)))
```

Figure three

Rule based systems may be tested and analyzed by placing tokens at the different Petri Net nodes. This corresponds to asserting certain facts in the rule based system.

Complex analysis of the rule based system is achieved by the extensive theory which surrounds Petri Nets, allowing one to determine, for example, if a set of markings is reachable and if the corresponding state possible.

The graphical representation of a rule based system may also help to decide what to abstract and modularize. When Petri Net components are abstractable to single nodes or to single transitions, the rules might be abstracted in a corresponding manner.

A Petri Net modeling capability has recently been implemented by the author using the Flavor System on the Symbolics computer. Petri Net models are being used to aid the development of a rule based prototype which manages and displays information in a Command, Control, and Communications system. Lessons from this prototype development will be presented at the symposium.

MODEL IDENTIFICATION USING HEURISTIC SEARCH

S. Hanagud and B. Glass
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia

1. Introduction

Control of distributed-parameter systems requires requires an accurate analytical model of the physical system. When the boundary conditions of the physical system vary unpredictably in time, as may be caused by growth, reconfiguration or damage, the problem of control of such a system becomes complex. Since control laws are based upon simulations of a system's behavior, and these simulations are obtained in turn from system models, the control of time varying distributed-parameter systems (such as robot manipulators or large space platforms) requires the repeated characterization of model structure to reflect the changes of the physical system. Within the domain of structural dynamics system identification, this paper discusses an approach to the model characterization problem, using heuristic search in a space of discrete system simulations configured as an inheritance lattice. An example from the domain of structural dynamics is used to illustrate the approach.

2. Problem Setting

Current system identification methods alone are inadequate for identifying time-varying models, as they start with a single static model structure. For example, suppose a parameter identifying program for an originally cantilever beam was given current dynamic response data corresponding to a new model, such as a clamped beam with a non-structural mass. The additional, unmodelled mass would effectively decrease the stiffness of the beam, which the parameter identifying program could accomodate only by removing stiffness from the elements of its cantilever model. It is apparent that parameter identification methods alone are not well suited to the model structure characterization problem. In order to obtain a model that matches a given physical system state, the characteristics of the structures of possible models must be known so that they may be compared with the evidence of the given state.

The structural dynamics theory embodied in each model was that of simple beams without significant shear or rotary inertia effects. The method of superposition of flexibility influence coefficients (Dunkerley's equation) was used to relate frequency shift to addition or movement of lumped point masses on a continuous beam. Node shifts were found by comparing the observed mode shape to the analytical mode of the same order associated with the selected model, and were used to estimate locations of lumped masses. Boundary conditions that varied bewteen each model resulted in a characteristic dynamic response (e.g., natural frequencies and mode shapes) for each model.

Characterizing the model structure will result in matching the i^{th} model out of n possible models. To completely identify the model, the i^{th} set of parameters must then be quantitatively identified by a parameter identification technique. This fully-identified model can be used for control and other applications, and is saved for monitoring and future identification reference.

3. Identification Procedure

The distinguishing characteristics of models that make identification possible, however, also foreclose the possibility of a unified analytical approach. In structural dynamics, for example, the model structure--the characterstic equations from which vibration frequencies and mode shapes may be calculated--varies by model. These variations take the form of changes in groups of boundary conditions and parameters (syndromes). Changes to an underlying conceptual model such as a beam suggests the representation of the possible models as objects in an inheritance lattice, allowing only the syndromes to be included at each node rather than repeating the entire model structure.

These distinct models are used to simulate the output of the physical systems they represent, so error criteria can be formulated and used in the search heuristics. Search is conducted in the space of

the simulations generated from each model. Two levels of abstraction are considered; first, prototype models representing general configurations, then more detailed models including the syndromes. The root node is the previous model. In structural mechanics, the first level examples could be various beam configurations (e.g., cantilever, simply-supported, clamped-supported, etc.), while the more detailed level could include the effects of added masses, stiffnesses, or cracks.

In an inheritance lattice, the detailed or "child" models can inherit structure and parameter values from the prototype "parent" models, with localized changes made at each node to distinguish it from its parent model(s). An assumption is made here that the syndromes are independent, that not more than one defined group of boundary conditions and parameters are affected by any one change to the physical system.

A tree of models in the domain of structural dynamics was constructed as a lattice of LOOPS [Bobrow 1981] objects, using simple beam models with syndromes of added masses at various locations. The search method used was best-first, with a weighted combination of squared frequency error and Allemang and Brown's modal correlation coefficient included in the model evaluation heuristic. A queue was formed, consisting initially of just the root node of the search tree (the simulation based on the previous model). If this first model in the queue satisfies the heuristic error constraints, the current model is retained. Otherwise, the children of this model are substituted for it, the queue is sorted by the least estimated error, and the process repeats recursively until a match is found or the bottom of the search tree is reached. Methods within each object, triggered by active values, correspond to discrete simulations. When triggered in the search process, these small attached programs calculated estimates of mass values and locations (given the current output from the physical system), as well as the vibration frequencies and mode shapes used by the searcher.

The identified model, including the initial parameter estimates of mass value and location, was given as input to a recursive parameter identification program, MCKID [Hanagud 1984] to quantitatively identify the model parameters. Parameter identification, following model structure characterization, completes the system identification process for time varying distributed-parameter systems.

4. Results and Conclusions

This model structure characterization approach was verified in the structural dynamics system identification domain by tests run with structural dynamics data input that reflected changes over time to a beam. Mode shapes and frequencies were obtained from analysis and corrupted with noise to simulate measurements (like those produced by a feature extractor, such as Spriet and Vansteenkiste's [Spriet 1984]) of a physical system's dynamic response.

It was found that even in the presence of significant noise, the search returned with the best-fitting model, in the relatively small search space considered. The existence of incomplete or missing data generally reduced the accuracy of initial parameter estimates, but did not affect the search--demonstrating robustness. Integration of this model structure characterization procedure with a parameter identification program resulted in the quantitative identification of model parameters.

Issues still unanswered include the optimization of the search heuristics for use in large model spaces and the presence of multiple syndromes simultaneously in the physical system. The addition of limited inductive problem-solving ability, and storage of previously-identified cases, would help relieve the latter problem.

References

Bobrow, D.G., and Stefik, M. (1981) The Loops manual. Technical Report KB-VLSI-81-13. Knowledge Systems Area. Xerox Palo Alto Research Center.

Spriet, J.A., and Vansteenkiste, G.C. (1984) Structure Characterization For Ill-Defined Systems. In T.I. Oren, et al. (Ed.), *Simulation and Model-based Methodologies: An Integrative View.* Berlin: Springer-Verlag.

Hanagud, S., Meyappa, M., Cheng, Y.P., and Craig, J.I. (1984) Identification of Structural Dynamic Systems with Nonproportional Damping. *25th AIAA Structures, Structural Dynamics and Materials Conference.* AIAA Paper 84-993. Palm Springs, California.

Using AI-Based Simulation to Model Manufacturing Facilities

Perry A. Zalevsky
ALCOA Laboratories
Alcoa Center, PA 15069

Computer simulation has been used to study the dynamic behavior of manufacturing facilities. In some applications it has been used as a design tool to examine proposed plant configurations. Issues relating to the placement and characteristics of machines and the processing paths for items are often of concern. Simulation has also been used to analyze possible changes in existing facilities. In these instances, the emphasis is on modeling an already existing system and varying parameters to determine their effect on that system. In this abstract I will briefly discuss applications in the two areas mentioned above and how AI is being combined with simulation to provide a more powerful analytical tool.

The design of new manufacturing facilities is a complex process that can be aided by the use of simulation. The number of machines, where to locate them, and the characteristics of each machine, are often items of interest. In a simulation, by varying each parameter, a large number of possible configurations can be evaluated. Other questions arise concerning items that move through the system such as the proper processing path, the amount of work-in-process inventory needed, and the means of transferring items between machines. Here again, the number of scenarios can be large. Also the processing costs associated with each of these alternatives may be important. Traditional simulation packages such as GPSS [1] and SLAM [2] have been and could still be used for this application. Since they were developed to be general-purpose simulation tools, they can model many different situations, including this one. But their generality makes them time-consuming and cumbersome to use. There also is no systematic way to reduce the number of scenarios generated by using the knowledge that is contained in each entity in the system.

An alternative approach is to use a frame-based AI knowledge representation scheme. In this way a library of entities can be created and instantiated, and can include both attribute and behavioral descriptions of each machine or item in the system. The knowledge base can consist of facts about each entity and its relationship to other entities, and process knowledge about the effect of actions in the system. For example, the class of mills with certain properties can be defined. Rolling mills can then be a subclass of mills, with the same properties as mills and any additional ones that are specific to rolling mills. A particular mill with values for each property can then be created. Each instance would then have all of the knowledge about itself and its relationship to all other entities in the system. It would then be easy to restrict the number of scenarios generated, for example, by cost, because the system-wide costs of any processing path would be easy to calculate. In this way the simulation could help choose the better alternatives.

We are using this approach to model manufacturing plants that ALCOA is considering building. At this time we are developing prototypes for two plants. The models are being developed on a Symbolics LISP machine using the SimKit knowledge-based simulation package.

The second kind of application that we are working on is the modeling of an existing facility. With the combination of AI and simulation, however, we are able to do new kinds of analyses. A number of our manufacturing plants are job shops where items move through the system while being processed at a large number of machines. Different kinds of items need different processing. Various machines are grouped together into work-areas where one person has the responsibility to schedule the processing of all of the items that come into that work-area. Over a number of years this person becomes an expert at scheduling this work-area. Each scheduler may have responsibility for 15 to 35 machines in a work-area. An item flowing through this work-area may need one or more processing steps, and can return after going to another work-area. The scheduler decides the path that the items will take, keeping in mind machine constraints, quality considerations, and most of all, due-dates. The scheduler may try to balance the flow of material over the machines, or route items around a machine if it is backed up. This expertise has been acquired over a long period of time and typically consists of many informal rules or heuristics.

The scheduling of items to be processed in a work-area is an expert system application. The expert could be interviewed, the heuristics determined, and possibly a rule-based system could be created to schedule like the expert does. But the combination of an expert system and a simulation would be much more powerful. Schedules generated by the expert system could be simulated and then checked by the expert. Once the system was validated then proposed modifications to machinery and processing paths, and more importantly, processing of new items never before attempted, could be off-line scheduled to see their impact on this work-area. This could all be done without affecting the actual manufacturing facility.

Another important use of such a system is for training. Expert schedulers acquire their expertise over many years. Quite often as they near the expert level, they also near retirement age. An expert scheduling system with a simulation would be an excellent tool to train new schedulers. As it is now, if an expert scheduler goes on vacation, performance of that work-area decreases noticeably in his absence. And in one particular case that we are working on, the expert scheduler is retiring in one and a half years. In the past, a novice who has spent a week or two with the expert would then take over this scheduling task. And the process of acquiring expertise would start all over again.

This work is also being done on the Symbolics LISP machine. It makes use of the SimKit package for the simulation and the KEE expert system tool for the scheduling system. Both of these packages use the same frame-based knowledge representation scheme and should be easy to put together.

References:

1. IBM, GPSS/360 Introductory User's Manual, GH20-0304-4, 1973.

2. IntelliCorp, The SimKit System Manual, 1985.

3. Pritsker, A. Alan B., Pegden, Dennis, Introduction to Simulation and SLAM, New York, NY: John Wiley and Sons, 1979.

4. Reddy, Y.V., Fox, Mark S., "KBS: An Artificial Intelligence Approach to Flexible Simulation", Technical Report, The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 1982.

ABSTRACT

REASONING ABOUT SIGNALS IN AN EXPERT SYSTEM FOR SIMULATOR ENGINEERS

by

Renato de Mori

and

Robert Prager

School of Computer Science
McGill University
Montreal, Quebec
Canada
H3A 2K6

The subject of our research is the problem-solving tasks of an engineer performing verification and tuning of mathematical models of real-time systems. Our goal is to produce an expert system which can identify and reason about discrepancies between a simulation and real-world data. The expert system must perform experiments with the model, trying to reproduce, within given tolerances, responses recorded during tests of the modeled system. Discrepancies in the results of an experiment lead to hypotheses about possible deficiencies in the model and corrections to the deficiencies.

We are applying AI paradigms to improve simulation, not by running an expert system as part of the simulation but by constructing a tool which can be used to improve real-time models.

The domain we have chosen is the aerodynamic models used in modern aircraft flight training simulators. The model is implemented as a program which is run at an iteration rate of, typically, 30 Hz. and is a critical component of man-in-the-loop simulation. The checkout data consists of time-histories; aircraft parameters, inputs and outputs, are plotted vs. time. Comparison of time-histories permits verification of transient and dynamic characteristics of the model. Interpreting time-histories is itself an interesting knowledge-based problem with parallels to speech understanding systems [1], [2].

The models in this domain are very complex and many details vary between different aircraft types. An experienced engineer has a mental model including terms common to all aircraft, equations of motion, effects of flight controls and expected response in standard manoeuvres. The engineer applies these general principles to understand a particular discrepancy in a particular manoeuvre. The reasoning tends to be qualitative [3] and concludes with a proposed modification to the experiment or model. Finally the modification is implemented and the experiment repeated to see the effect on the discrepancy. Our expert system must perform the same basic cycle:

- run experiment
- interpret results
- analyze discrepancies
- modify model/experiment

This is essentially a search through a space of possible models and experiments, but the nature of the problem forces us to use advanced AI techniques.

Since rules for modifying the model or experiment cannot be exhaustive it is essential for the system to learn how to make better modifications by examining the difference between predicted and actual effects as experiments are repeated. This form of learning by trial and error takes place among human experts also.

The expert system begins its task with a set of manoeuvres for the model to reproduce. A plan must be formulated to control the sequence of experiments. If the model is changed to correct a particular discrepancy other manoeuvres may be affected, this constrains the allowable modifications. The control problem of revising plans to repeat earlier manoeuvres requires planning with constraints [6]. Other activities take place in addition to planning and the basic cycle mentioned above. For example it is necessary to look for sets of minor discrepancies which together provide evidence suggesting an unacceptable deficiency in the model [4]. With many cooperating processes the appropriate paradigm is the blackboard architecture for control [5].

Advanced frame-based representation techniques are used. The blackboard is divided into control and domain knowledge bases. The domain KB is further divided, containing knowledge about the aero model and results of all experiments. These KB's

are accessed by multiple processes, messages are sent between frames and multiple contexts are needed to represent the results of different model modifications.

All the elements described so far appear in the sub-problem of curve interpretation and most of our work has concentrated on this area. The goal of curve interpretation is to find discrepancies between aircraft and simulator time-histories. A time-history is a sequence of values of a parameter, equally spaced in time. For each curve a separate process is created, each process calculates a polynomial approximation to its time-history. The order of activation is knowledge-based. The required accuracy of the approximation is expressed as a constraint which the process must satisfy. Advantages of using polynomials include filtering noisy data, easy comparison of curves and easy calculation of derivatives.

The constraints on accuracy of approximations depend on the parameter, manoeuvre and reasoning in progress. If an approximation is not sufficiently accurate the process can add new breakpoints or higher degree terms. If an approximation barely satisfies its constraint it may be necessary to tighten the constraint so other approximations which depend on the first will be accurate enough.

Our first aero model has three degrees of freedom. In this case only three approximations - airspeed, altitude and pitch angle - need be computed numerically, the rest can be obtained using the equations of motion and polynomial arithmetic. Aircraft curves are, in general, only approximated once. Knowledge about the polynomials for aircraft curves is applied to help determine polynomials for simulator curves.

Each manoeuvre is a sequence of phases, in each phase different aerodynamic effects predominate. Phases are determined by significant movements of flight controls, entering/leaving ground effects, stall etc. A knowledge source provides generic descriptions of the phases of all manoeuvres, including expected shape of input and output curves. Interpretation processes use knowledge about phases to pick breakpoints and fit a seperate polynomial in each segment.

Once accurate approximations for simulator and aircraft curves are known these can be compared and discrepancies found. The presence of a discrepancy triggers further curve interpretation which depends on the discrepancy and the qualitative relations used during reasoning. Acceptability of discrepancies is also context-dependent. For example an error of ten feet in altitude is acceptable in a climb test but not in a landing. One important type of error is incorrect latency. This is when the shape of the simulator's response to an input is correct but occurs too early or too late. The system must try shifting, by time Δt, the simulator curve to best match the aircraft curve and then reason about Δt.

In conclusion, we have investigated a complex problem of the interaction between simulation and expert systems. We have applied emerging techniques such as knowledge-based scheduling, cooperative problem solving and qualitative reasoning to a real world problem. Experience in knowledge-based interpretation of signals in speech understanding research has been profitably applied to the interpretation and comparison of parameter histories.

REFERENCES

[1] De Mori, R. and Laface, P.
An expert system for speech decoding
Proc. AAAI Conf. Pittsburgh, PA, pp. 107-110, 1982.

[2] Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R.
The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty
ACM Comput. Surv., Vol. 12, pp. 213-253, 1980.

[3] Forbus, K.D.
Qualitative Process Theory
Artificial Intelligence, Vol. 24, No. 1-3, pp. 85-168, Dec. 1984.

[4] Gordon, J., and Shortliffe, E.H.
A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space
Artificial Intelligence, Vol. 26, No. 3, pp. 323-357, July 1985

[5] Hayes-Roth, B.
A Blackborad Architecture for Control
Artificial Intelligence, Vol. 26, No. 3, pp. 251-321, July 1985

[6] Stefik, M.J.
Planning with Constraints
Stanford Heuristic Programming Project, Memo HPP-80-2,
Computer Science Department, Report No. STAN-CS-80-784, Jan. 1980.

5/27/86

KNOWLEDGE-BASED SIMULATION

Jeff Rothenberg

The Rand Corporation
1700 Main Street
Santa Monica, CA 90406
213/393-0411

(Abstract for AAAI Workshop on AI and Simulation 8/11/86)

The Knowledge-based Simulation project at Rand combines simulation and reasoning in an attempt to solve severe deficiencies in large-scale military simulations. It combines object-oriented simulation with expert systems techniques, emphasizing hybrid representation, simulation at multiple levels of abstraction, and graphic explanation and exploration.

Background

The seminal work of Newell, Shaw and Simon at Rand in the 1950s dominated much of AI's early research and defined many of its continuing focal points. In the last decade Rand research on expert systems has produced the RITA and ROSIE languages, as well as several expert system applications. Simulation research at Rand produced the SIMSCRIPT language as well as theoretical and experimental research in game theory, monte carlo simulation, and military wargaming.

More recently our simulation research has synthesized ideas and techniques from artificial intelligence, expert system technology, graphics, and distributed computing. Key results include the object-oriented simulation language ROSS that makes simulations easier to build and maintain, the Time Warp technique that reduces the execution time of object-oriented simulations by using a network of processors, and two prototype simulations in ROSS: the SWIRL air battle simulation and the TWIRL land battle simulation.

SWIRL and TWIRL (like most simulations) do not include a model of the domain. They relate situations to actions with no description of why the actions are necessary, no representation of objects' motivations or intentions, no specification of what should happen if actions are taken without all of their conditions being satisfied, and no depiction of the reasoning process that chooses actions. Simulation languages rarely provide mechanisms for encoding and using such knowledge: they force the reduction of goals and beliefs to opaque situation-action rules.

This severely restricts simulations to answering "what if" questions, where the simulation determines future states given an initial situation. Without an inferencing capability, simulations cannot specify what objects should do to accomplish some specified goal, which future states cannot possibly exist, what alternative actions might improve an outcome, etc. Such questions can only be answered by systems embodying descriptive models of the domain such as those found in Knowledge-based (expert) systems.

Extended Modeling Paradigm (Hybrid Simulation)

We are augmenting the object-oriented ROSS language in several ways. Objects will retain most of their current ROSS characteristics, including multiple hierarchies for the inheritance of attributes and behaviors. In previous

ROSS simulations generic ("class") objects do not themselves respond to messages; all messages are received and responded to by instance objects. We are experimenting with allowing simulations to vary their levels of abstraction and aggregation by associating behaviors and attributes with generic objects which simulate the behavior of their instances and maintain attributes representing aggregations of the attributes of their instances.

We are exploring alternative rule paradigms to represent objects' behaviors, intentions and reasoning processes as well as the behavior of the simulation itself (e.g., selecting appropriate levels of abstraction and aggregation based on the user's stated needs). Our goal is to satisfy the dual criteria of making simulation code easier to understand and amenable to automatic inferencing.

In integrating these ideas, we intend to use rules and constraints to separate those aspects of a simulation that are really descriptions of the simulated world from the objects of interest in that world. Much of the physical, logical and temporal information in a simulation embodies the fact that the real-world environment behaves according to certain laws. As such, its relevance to the model is only that it must provide a valid simulated world in which the model can run. The appearance of such information in a simulation is a distraction that makes it harder to see that part of the model that is of interest.

Similarly, aspects of a simulation such as automatic unplanning, control of inferencing, or control of level of aggregation, and "artifactual" aspects such as those controlling graphic presentation and interaction are also irrelevant to the model itself. In all such cases we are experimenting with declarative forms to separate such knowledge from the model of interest.

Intelligent Explanation

Explanation requires that a simulation keep track of what it has done and be able to analyze its own execution history and behavior specifications, presenting this analysis to the user in understandable form. The system must maintain an execution history of events that have occurred, rules that have been invoked, messages that have been sent, prior values of attributes and states of databases, magnitudes of changes, etc. We are experimenting with various representations of simulation history for producing "execution trace" style explanation.

The primary task of explanation is to convince the user that a model is behaving reasonably, and to show how it arrived at a particular result, as requested by the user's stopping the simulation interactively and indicating (graphically or by means of a query) the result that is to be explained. The user must also be able to back up to a previous point in the simulation, since a key result may not be recognized until after its occurrence.

We are developing a graphics facility for performing such interactions, emphasizing the ability to animate selected portions of a simulation. We plan to give control to the system (via rules) and/or the user (via direct interaction) over the level of graphic abstraction presented, so as to minimize visual clutter and display aggregated results.

Intelligent Exploration

Exploration allows a user to selectively modify a simulation, pursue excursions, focus attention on selected aspects of the model, perform sensitivity analysis or ask how particular results might be achieved.

The graphic interaction described above allows the user to select objects graphically and edit their attributes or behaviors explicitly. This

provides a natural way to specify a scenario and set up initial conditions for a simulation run. We also plan to experiment with using graphical input to specify procedural information, for example allowing the user to draw a route for a moving object on the screen. The system would capture (and generalize) the relevant information, representing it as a new behavior for the object. This "graphic behavior modification" should allow non-programmers to specify simulation scenarios more easily.

Similar techniques will allow interrupting a simulation and trying alternative excursions, selectively modifying attributes and behaviors. We are experimenting with alternative approaches to relaxing constraints during such explorations, under user control.

Another major concern is to allow analysts to perform sensitivity analysis on a model to identify important factors. We are examining both static and dynamic approaches to this problem. Static approaches include applying inferencing to rules and logical constraint declarations, or applying analytic methods to closed-form mathematical constraints. Dynamic approaches include automatically generating and running excursions to perturb selected variables. In both static and dynamic cases we are experimenting with a hierarchical representation of sensitivity to facilitate complicated analyses.

Finally, we are examining the utility of goal-driven simulation, where the user specifies a hypothetical result and the system tries to find a way to achieve it by a combination of static analysis of constraints (to eliminate certain results as impossible), forward chaining from a given set of conditions, and backward chaining from desired goals or hypotheses.

Palladian Software: Wrapping AI techniques around simulation and other Operations Research tools to help manufacturing managers.

May 29, 1986

Palladian Software's mission is to create software products which make the quantitative and qualitative techniques of management science accessible and useful to ordinary managers. The expertise engineered into our systems by our Technical and Product Management teams is extracted both from professors at MIT's Sloan School of Management and from managers at the Fortune 500 companies which are participants in our innovative "Development Partner" program.

Palladian's first announced product, the Financial Advisor™, is an expert system for doing sophisticated financial analysis of business decisions. It was commercially released in April. A second product, which is still under development, is an expert system for evaluating different manufacturing policies and is called the Operations Advisor™. This abstract describes the Palladian Operations Advisor™'s ("POA") functionality for the first and future releases.

Manufacturing software traditionally attempts to track all of the complexities of the factory. However, this forces the manager to focus primarily on the details. The purpose of the POA is to help high level manufacturing managers make proactive, rather than reactive, manufacturing policy and capacity planning decisions. The POA "understands" the factory not in terms of the details of a snapshot of the operation, but in terms of flows through the factory. By using the POA's "understanding" of the "physics of manufacturing" as manifested through such techniques as simulation, the POA can predict these flows from a simplified description of the factory. The POA thus encourages the manager to step back, look at, and manage the factory through the allocation of capacity and the setting of policies.

In advising on this domain, the POA uses a model developed by a team lead by Gabriel Bitran, the head of the Operations Management Department at MIT's Sloan School of Management. The model incorporates several core concepts: queuing networks, incremental costs, heuristics for determining tradeoffs among various important measures, and the recognition of patterns of information useful in making manufacturing policy decisions. The POA also "knows" a set of methods developed by

Richard Berenson [59]
Pallaldian Software Inc.
Four Cambridge Center
11th Floor
Cambridge, MA 02142

the team for using the model to analyze the factory.

Initially, the POA will only advise on a restricted domain of factories: discrete, fabrication processes. Assembly operations which share the same "physics" as fabrication operations (e.g. circuit board assembly) will also be covered. Assembly which involves the joining of major sub-assemblies or frequent stockouts of required parts will not be covered. Continuous processes which cannot be viewed as a single step in a discrete process or for which the continuous nature of the step is important in understanding the process will also not be covered.

These POA restrictions reflect our development philosophy: the basic capabilities provided by the system will create significant value so that it is not necessary to provide nearly the complete functionality of the system all at once. Rather, with each release, the system will grow in terms of the domain it covers, the techniques it incorporates, and the ability it has to help the user to use those techniques.

Technical Overview

The technology at the heart of the POA is "model-based reasoning." Within the model of the factory are sets of objects representing work centers, product families, operations, and so on. Each object has a frame of slots; the slots within and between frames are tied together by a network of constraints and relationships. The relationships among the objects themselves are represented both in the structure of the database and in other objects through such things as collections and sequences.

Information in the slots can be user entered through "windows" into the objects, can be inherited through context-dependent defaults, or can be computed by "demon goals" such as constraints (see e.g. Steele, 1980) or single directional functions.

The POA uses two related models from Operations Research to estimate the average lead time and work in process inventory levels in a factory given described policies and capacity levels. The first, analytical queuing theory, can be viewed as a

heuristic approximation to the second, monte carlo simulation. The queueing approximations run several orders of magnitude faster than the simulation and are highly accurate for virtually all queueing networks. Simulation, however, is more flexible and can be made to model details in the operation of the factory which cannot be represented in the queuing approximations.

Our approach to using these Operations Research techniques, such as simulation, is to build a buffer of translation between the user and the technical details of the technique. The buffer has several components. The first is extensive use of interactive graphics. For example, the user describes the factory by graphically placing work centers and graphically routing product families among them. The second is aid in formulating how the techniques should be used. For example, the user might be asked, based on an automatic analysis of the number of hours of capacity at each work center, whether work centers have significantly different schedules. If they do, the user might be asked to describe the schedules for work centers where they are not already described. The simulation would then know to model, rather than ignore, those schedule differences.

The third element of our approach to techniques like simulation is the interpretation of results. For example, if the user has indicated a concern about jobs occasionally being very late, the system would automatically track, compile, and interpret statistics on the distribution of lead times. The interpretation would be not in technical jargon, but in terms which relate directly to the manufacturing context. In this way, we can make the capabilities of simulation accessible and understandable to an audience which might not otherwise be interested in or able to take advantage of them.

In addition to these methods of making the analytical techniques available to managers, the POA has features which help people to use the system itself. For example, there is an extensive scenario management system which lets users create, store, and compare a variety of assumptions about the factory.

The POA is a composite expert system. The system makes limited use of production rules for some functions, such as consistency checking, but it relies mainly on other AI technologies such as model-based reasoning, object-oriented programming, goal-directed control, plans, and constraint-based inference. Palladian takes advantage of these technologies to accomplish our goal of creating a high value system to make the techniques of management science available to everyday managers.

APPLICATION OF THE BUTTERFLY PARALLEL PROCESSOR TO THE SIMULATION AND CONTROL OF COMPLEX SYSTEMS

David M. Fram

BBN Laboratories Incorporated
Cambridge, MA 02238

The Butterfly Parallel Processor is composed of processors with memory and a multistage switching network that interconnects the processors. A Butterfly system can be configured with from 1 to 256 Processor Nodes. Each Processor Node consists of a processor, memory, and an interface to the switch. Collectively, the memory of the Processor Nodes forms the shared memory of the machine. All memory is local to some Processor Node; however each processor can access any of the memory in the machine, using the Butterfly Switch to make remote references. (From the point of view of an application program, the only difference between memory on its local Processor Node and memory on other Processor Nodes is that references to remote memory take a little longer to complete.) The speeds of the processors, memories, and switch are balanced to permit the system to work efficiently in a wide range of configurations.

Each Butterfly Processor Node contains a Motorola MC68000 microprocessor (or a MC68020 with MC68881 floating point co-processor), at least 1 megabyte of main memory, memory management hardware, an I/O bus, and an interface to the Butterfly Switch. I/O connections can be made to each Processor Node, making I/O configuration very flexible.

The Butterfly Parallel Processor can be programmed in C, Fortran, and multi-processor Lisp. An Expert System environment is currently being developed on top of the Lisp.

Developed as a research vehicle under sponsorship of the Defense Advanced Research Projects Agency, Butterfly by now represents a capable parallel processing technology suitable for wider commercial use as well. In considering commercial product directions for the Butterfly Parallel Processor, BBN is currently focusing on two major application domains: 1) simulations of complex systems; and 2) ambitious real-time systems for monitoring and control. In the remainder of this abstract, we describe these domains further and explain how the Butterfly architecture is well suited to them.

As physical systems grow ever more complex, simulation is becoming more and more important as a means of gaining insight into the behavior of these systems under both normal and abnormal conditions. Simulation can have very high economic value, as when it alleviates the need to build and test prototypes. Simulation can also make it possible to explore scenarios that would be impractical to carry out in the real world, such as simulating the results of placing an overload on an electrical network.

In addressing the simulation market with Butterfly, BBN will concentrate on applications that exhibit one or more of the following characteristics:

- the system being simulated is large and complex; or
- the component elements of the simulation have a direct physical

interpretation and are individually complicated. (For example, in a battle simulation, the elements might be the ships, planes, and submarines taking part in the battle. In a chemical plant simulation, the elements would be the various pieces of equipment in the plant.); or

- there is a demanding real-time requirement on the simulation.

Applications exhibiting these characteristics map extremely well onto the Butterfly architecture. Large, complex systems can make good use of the Butterfly's large amount of memory and shared memory architecture. Systems involving identifiable component elements parallelize easily; one simple conceptual model that can be applied is that each physical element is represented by a different processor in the machine. Simulations with real-time requirements benefit directly from the ability to scale up to a very large number of processors; performance can be guaranteed by allocating a sufficient number of processors to the problem.

The second targeted application area is sophisticated real-time monitoring and control. An illustrative example is a satellite control facility under development by a major aerospace contractor. This system will monitor telemetry data from satellites, and will use a combination of artifical intelligence and simulation to permit system operators to analyze and control the internal systems in the satellite. The ability to handle high-speed input, to combine artificial intelligence with conventional programming, and to be able to scale the system up to provide real-time response are all important.

The Butterfly offers significant advantages for performing applications in this area:

- The applications frequently interact with the real world with tight time constraints, and the Butterfly architecture and operating system are well-suited to maintaining guaranteed real-time response.

- The applications tend to be I/O-intensive as well as compute-intensive, and the Butterfly's I/O bandwidth can be incrementally expanded to very high levels.

- The applications typically require a mix of real-time processing, numeric processing, and symbolic processing; the Butterfly is perhaps the only existing parallel processor that can support computing environments appropriate to all of these.

The areas just described--complex system simulation and large-scale integrated systems for monitoring and control--are closely inter-related. In developing a large-scale control system, simulation is often required as a technique for understanding the system to be controlled and for testing and validating the resulting control system. In operating a large-scale control system, simulation is often used as a technique for predicting the likely outcome of some contemplated control action. For example, in a plant control application, in order to decide upon the optimal control strategy under a given set of conditions, it might be reasonable to "try out" several different strategies in a simulated mode to see which strategy produces the best results. For this to be useful, the simulations must be highly accurate (the underlying models must be complex enough to capture what really happens) and they must run fast enough so that a number of different simulations can be carried out in the time allotted for making a decision. The requirement to have complex models that run extremely rapidly places demanding performance requirements on the simulation system and motivates the use of parallel processing techniques.

61

Knowledgebased Interface to Process Simulation

Roar Fjellheim, Einar Dehli, Ulf Wretling, Kjell Tangen, Per-Olav Opdahl and Odd-Wiking Rahlff
Computas Expert Systems
P.O. Box 410, 1322 Høvik, Norway

1. Introduction

Design and use of simulation models for complex industrial plants is a demanding and time-consuming task, requiring extensive knowledge of both the plant and the simulation system. While the basic principles and tools for process flowsheeting (calculation of steady-state heat and material balances in chemical processes) are well understood (e.g. [1]), there is a definite need for improved man-machine interfaces and support systems for exploiting the powerful flowsheet simulator programs now available.

This article describes a project that develops an intelligent frontend for the **PROCESS** flowsheet program [2]. The KIPS frontend (Knowledgebased Interface to Process Simulation) is being implemented using **Loops**. The project is conducted under a contract with Statoil, Den norske stats oljeselskap A/S.

The work pattern with KIPS will look as follows. The engineer with some initial design ideas starts by translating these into a flowsheet on the graphics screen of the KIPS workstation. The design work is supported by knowledge embedded in the KIPS database, which contains basic building blocks and model (sub)assemblies, as well as design rules. As soon as the model has been constructed in an interactive session, the corresponding input file is automatically generated and sent to PROCESS for execution. The advantages are:

- The knowledge involved in designing a model is shared between the engineer and the system. KIPS may serve as a repository for good design practise, and for accumulating experience.
- The engineer is freed from manual handling of drawings, preparation of detailed input files, etc., and may concentrate on high-level issues of flowsheet design instead. The user interacts with the system in terms of familiar process symbols.

2. System structure

The four major components of KIPS, as shown in Figure 1, are:

1. The objective of the system is to support design of **Process Models**, which are input from, and displayed to the user graphically as process flow diagrams, and converted to text form for PROCESS.
2. The process models are created and manipulated through a number of interactive **Design Functions**. Foremost among these are a graphical editor for flow diagrams and a PROCESS input file generator.
3. The design work is supported by one or more **Knowledge Base(s)**, where PROCESS concepts and previously acquired model components, design rules, etc. are encoded.
4. The user gains access to the remainder of the system through a uniform **User Interface**, inspired by the Xerox Star [3] design.

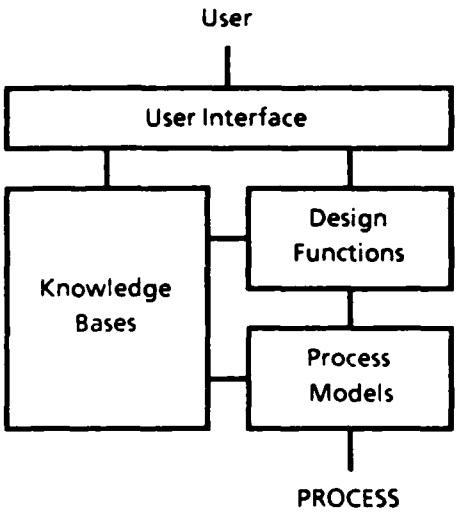


Figure 1. KIPS main modules

The knowledge base is a major focus for this work. Constructing a PROCESS model is largely seen as copying pieces of information from the knowledge base into the evolving model, then modifying the pieces to suit the current needs. This instantiation of general descriptions to particular instances is accompanied by activation of design rules for checking the validity of user actions, providing advice upon request, or automatically filling in parts of the model.

3. Knowledge base

The "backbone" of the KIPS knowledge base is a class hierarchy defining model components and topology, augmented with rule sets as described

in more detail below. Figure 2 depicts a central fragment of the class structure:

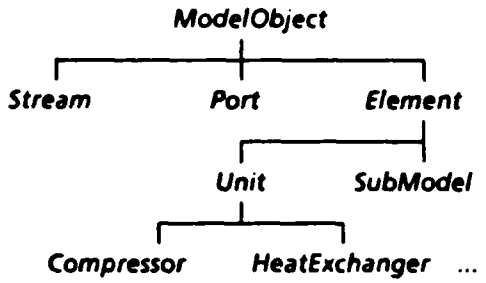


Figure 2. Model object taxonomy

A model contains zero or more **ModelObjects**. Each ModelObject is either an **Element**, a **Port** *or* a **Stream**. The Element is a general class, with subclasses **SubModel** (i.e. a submodel with its own internal structure), or a basic **Unit**, of which **Compressor**, **HeatExchanger**, etc. are specific subclasses. The units (e.g. Compressor) may spawn further subclass definitions, corresponding to particular ways of using these units (based on experience). Figure 3 shows some of the currently defined PROCESS units. The inclusion of both SubModel and Unit as building-blocks in models means that fully recursive structures are possible.

Each unit has a number of parameters that can be specified to PROCESS. Each parameter is described by an IV having a number of properties:

type	parameter type: number, text, reference to unit, etc.
default	default parameter value
unit	the engineering unit in which a numerical parameter is given
range	minimum/maximum values
values	list of legal values
reason	why does the parameter have its current value (a text)

etc.

A specification of a unit is defined as a particular way of using that unit, minimally defined by partitioning the set of parameters for the unit into required, optional, special, fixed and unused parameters. The specification of a unit may change during a session, and the system will change its behaviour with respect to that unit correspondingly.

We now describe a common framework for constructing (usually small) rule sets associated with particular parameters. Each unit parameter description may have associated ruleset methods of the following kinds:

check	method for checking validity of a proposed new value

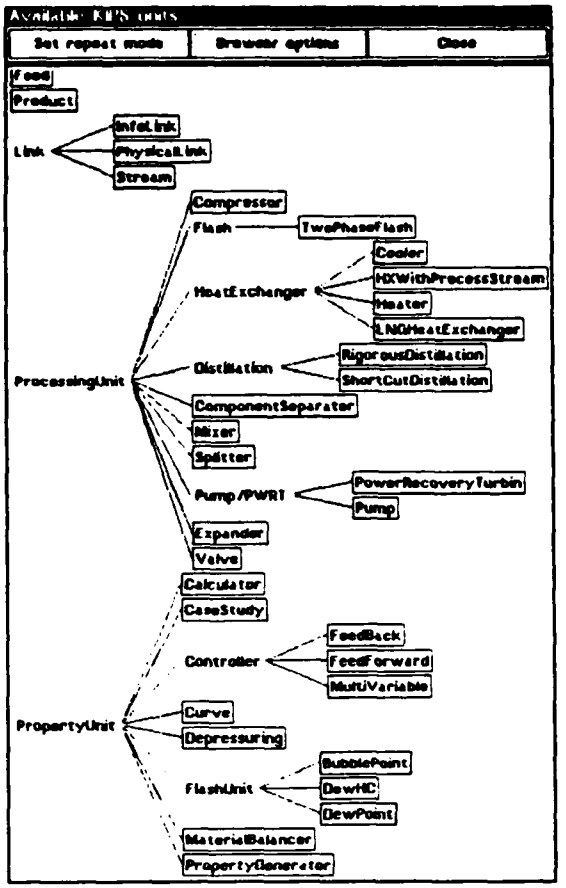


Figure 3. PROCESS unit taxonomy

advice	method for advising the user on how to select a new value

The rulesets are activated as side-effects of reading and updating the parameter value:

- Before an attempted update of a parameter value, a ***check*** method runs, and if the value is invalid, an error message is displayed to the user.
- The user may ask the system for ***advice*** about how to specify a parameter value. A method associated with the parameter will then be activated.
- If a parameter has no specified value, a ***get-function*** will be activated which will look for a default value, when needed.

AI programmers will recognize this schema as an example of the use of "demons" in frame systems.

Given the facilities described above, "knowledge programming" [4] in KIPS takes two principal forms:

- Refinements of the **classification** hierarchy, e.g. defining subclasses of an existing unit

class to capture different idiomatic uses of that unit.

- Definition of rulesets associated with particular parameters. In the KIPS system, there are hundreds of such rulesets, each typically containing less than ten rules.

The maintenance of the knowledge base should not be left to the users, but be entrusted a **knowledge base administrator** (who may at the same time be a user). Figure 4 illustrates the fundamental work cycle of KIPS : Users build process models by instantiating knowledge base entities, while the knowledge base administrator inputs basic PROCESS definitions, as well as generalizes particular models (model elements) into the knowledge base.

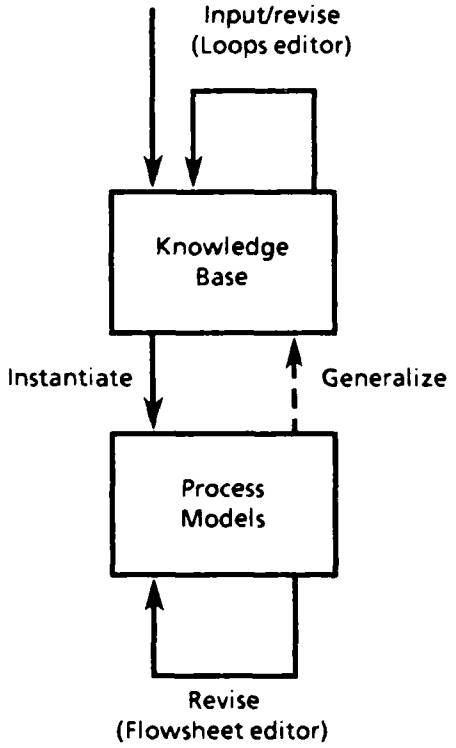


Figure 4. Knowledge instantiation/generalization

4. User interaction

When a user starts up the KIPS system, the screen will contain a number of small **icons** [3] representing **knowledge bases** and **flowsheets**. By pointing at an icon with the mouse and clicking a mouse button, the icon will open up to reveal its contents. Knowledge bases are displayed as LOOPS classification graphs, while flowsheets are diagrams closely resembling those used by process engineers. Figure 5 shows a snapshot of the KIPS screen as it might appear to an engineer designing a PROCESS model.

A new unit or submodel is placed in a flowsheet by pointing to its name in a knowledge base, and then point to a location in the flowsheet where the element should be placed.The unit is moved around by animation. Then, parameters may be filled in (via "pop-up" menus), and the element connected to other flowsheet elements by pointing. The rulesets come into action during this process, as described earlier. Finally, a menu attached to the flowsheet may be activated for producing hardcopy output or generate an input file for PROCESS.

Hierarchical design is a most important aspect of design processes, and is supported by the general model structure shown in the previous section. Functions for creating submodels, zoom in on the internal structure of submodels, etc. are available to the user. We believe this is a fundamental requirement for any system purporting to support design in a complex domain, such as simulation modelling.

5. Current Status

Today, in early may 1986, a prototype of KIPS is in use by the research and development department at Statoil, mainly for demonstration and testing purposes.

- The graphical flowsheet editor is operational but further enhancements are under development.
- PROCESS units regarded as relevant for offshore simulations have been implemented.
- PROCESS is a sequential simulation program and a precedence ordering is suggested to the user, using Gundersen's algorithm [5,6]
- An input file is automatically generated and transmitted to a host computer (an IBM main frame) and the simulation results are transmitted back to the Xerox workstation.

6. Further work

A major motivation behind the KIPS project is finding ways of including *design knowledge* in the intelligent interface to PROCESS. This is clearly a vast topic, and we have chosen to approach it in an incremental fashion.

One way of viewing the organisation of process modelling knowledge is related to the hierarchical structure of models:

Process structuring

On the highest level, the overall structure of the process is selected. Relevant constraints are the goals of the process (e.g. in terms of external feed/product properties), previous design experience, safety standards, etc.

Submodel/unit selection

Given the overall model structure, the next step in the refinement is to choose particular submodels/units to fill the structure.

Parameter selection

Finally, the parameter values of the individual units must be set so as to achieve the desired role of the unit in the overall process.

Logically, the design should proceed top-down, and the reasons for decisions at different levels should be maintained for later project phases. Unfortunately, this is seldom carried out in practise, thereby losing a major source for making informed model revisions.

Another possible direction for the KIPS project is to extend the present concentration on the model design aspect to also include other phases of simulation projects:

- Planning (sequences of) simulation experiments to achieve given project goals.
- "Qualititative"/coarse simulation in the frontend to weed out first-line problems.
- Fault diagnosis of simulation if the result are wrong or non-satisfactory.
- Evaluation, interpretation and presentation of the simulation results.

References

[1] A.W. Westerberg et al. : *Process Flowsheeting*, Cambridge University Press, 1979.

[2] N.F. Brannock et al. : *PROCESS^{SM} Simulation Program, A Comprehensive Flowsheeting Tool for Chemical Engineers*, Computers & Chemical Engineering, Vol. 3, pp. 329-352, 1979.

[3] D. C. Smith et al. : *Designing the Star User Interface*, BYTE, April 1982.

[4] M. Stefik et al.: *Knowledge Programming in Loops*, AI Magazine, Fall 1983.

[5] T.Gundersen & T Hertzberg: Partitioning and Tearing of Networks - Applied to Process Flowsheeting, Model, Identification and Control, 1983, 139 - 165.

[6] T.Gundersen & T. Hertzberg: Notat: Beregnings-rekkefølge og tearing, Trondheim, 1984.

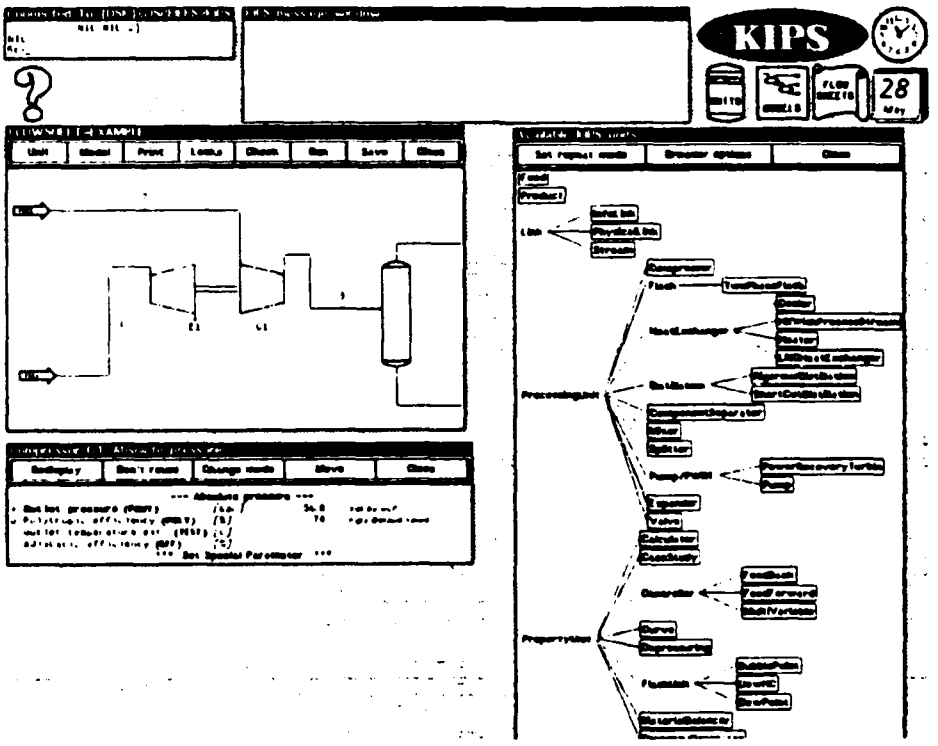


Figure 5. KIPS screen snapshot

Accepted at ECAI-86 (Brighton, UK)
62

THE ANALOGICAL REPRESENTATION OF LIQUIDS IN NAIVE PHYSICS

Francesco Gardin*,**, Bernard Meltzer*, Paolo Stofella*,**

*AI & Robotics Laboratory, JRC-Ispra, 21020 Ispra, Italy
** Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Moretto da Brescia, 20100 Milano, Italy

Abstract

The modelling of our intuitive knowledge of everyday physical behaviour of liquids is unlikely to be successfully achieved by the use either of classical hydrodynamics or current qualitative reasoning approaches. The use of analogical representations for the purpose has been studied, in which liquids are represented as composed of base elements of pixel aggregates of a graphics system. Computationally speaking these "molecules" are actors, the intuition of causality being realized in the constraint that only adjacent ones can exchange messages. Eight rules (embodying fundamental properties of fluidity, gravity and causality) which govern these exchanges suffice for the generation of qualitatively correct behaviours in a wide variety of situations. Limitations of the present stage of development of the model, and issues involved in incorporating such models in problem-solving systems are briefly referred to.

1. Analogical representation

The results to be reported here are part of an ongoing investigation of the use of analogical representations in modelling naive physics (**Gambarella et al. 1986**). What one is looking for are computational models of the everyday behaviour of liquids familiar to everybody, such as : what happens when a glass of water is tilted, how the level of water changes when it is poured into a container, how the water would spread on the floor if the bucket leaked, splashing, and so on.
It seems unlikely that much success would be achieved if one tried to design such models on the basis of the equations of classical hydrodynamics (**Feynman 63**). Not only because of the extreme complexity of such equations and their boundary conditions in non-trivial situations, but also for other reasons : Firstly the concepts and terms used in such representations are far removed from notions used in commonsense reasoning, such as : assuming the shape of a container, being contained by, dropping, splashing, etc. Secondly, the behaviours predicted by such models are far more quantitatively detailed than the rough estimates that suffice for normal everyday and engineering use, and in addition it is often very difficult to extract from such "exact" solutions intuitively simple properties of the behaviour.
In recent years therefore studies have been initiated in the field of qualitative physics aimed at developing more suitable models of our intuitive knowledge of the physical world ((**Bobrow 84**) and for liquids (**Hayes 79b**), (**Forbus 84**)). The cited studies on liquids are fragmentary attempts to axiomatize such knowledge, without however having reached the point of implementation in programs. It is difficult to predict how successful such an approach may be, but one or two remarks are in order. One's impression is that a reasonably complete axiomatization would be very large indeed, and its implementation in programs would come up against all the usual control problems of systems of deductive inference as well as the frame problem (**Hayes 73**). Also, as pointed out by Sloman (**Sloman 84**) such formalisms involve strong ontological commitments, that is, fixed choices of entities and concepts to be represented.
Both of the representation schemes discussed above are of the type Sloman (**Sloman 71**) termed Fregean. In Fregean representations everything, essentially, is represented in terms of one basic structure, namely the application of a function symbol to one or more argument symbols, whatever

may be the subject matter ; predicate logic is of course the paradigm example, but much of mathematical, computing and natural languages is also of this type. This means that in such cases there is no or little structural similarity between the representation and what is represented. Representational schemes in which such structural similarity predominates were termed "analogical" by Sloman (**Sloman 71**). Examples are maps, diagrams, flow-charts.
The analogical representation of liquids used in the present work is a two-dimensional one taking the form of sets of pixels of the two-dimensional array of a computer graphics system (The issues of three-dimensional modelling have not yet been taken up, though they do not seem to be fundamentally different from those studied). In this way spatial properties and relations of the liquid systems are already implicit in the model itself. It will be seen that the operations required to give qualitatively correct behaviour of the liquid model are relatively few and simple.

2. The methodology used

Since in the ordinary behaviour of liquids different parts of a given mass of it have different motions, our representation evidently must make use of decomposition, that is, the whole mass is represented as the sum of its component parts. And since we do not know in advance at what point in the liquid mass this differentiation of states of motion will occur, we are compelled in our analogical representation to choose some basic grain size for the decomposition. That is to say, our model, graphically speaking, will be a configuration of identical basic elements, each of which is an aggregate of pixels, of some convenient shape, e.g. circular or square. The basic operations of the model then will be on these elementary "molecules".
Next, in accordance with our intuitive notions of causality, we limit direct interactions between molecules entirely to those between spatially adjacent ones. Thus all the rules of behaviour will be local ones, whose joint operation gives rise to the global behaviours. Quite apart from the question of causality, it is probable that if we had tried to develop a system of global rules instead, it would have been overly complex both in respect of its size and of the conditions to be attached to each rule, to cope with the great variety of macroscopic behaviours possible.
These two requirements of molecular decomposition and localness of rules almost determine the appropriate style of programming to be used. That is to say, the molecules, computationally speaking, will be actors, with adjacent ones interacting by exchance of messages, such exchanges of course also occurring with pixel actors of the environment like those
in the representation of containers.
And indeed with the availability of a Symbolics 3600 Lisp machine with its "flavour" actor facilities, this object-oriented programming style was chosen, also because of its architectural similarity to the operation of massively parallel computers (**Hillis 85**), which, when they become available, will be the ones most suitable for implementing analogical representations.

3. Rules of liquid behaviour

Analogical models of physical systems may be used for various purposes, such as envisionmentoftheir functioning, planning, looking retrospectively at their behaviour in fault-tracing, as psychological mental models, as components of explanation systems, as parts of problem-solving systems, etc., but central to the applications is their capacity for simulation of behaviour, and it is on this aspect that this research has been concentrated.
In the following summary of the local rules which govern the exchange of messages between adjacent actor molecules, we term a molecule free if the space immediately underneath it is not occupied, and constrained otherwise.

1 A molecule can change its position to a neighbouring one if that one is not occupied by either another molecule or a rigid body (Non-copenetrability)
2 A molecule can receive messages and pass them on to its neighbours (Causal transmission)

3 A free molecule moves down until either it encounters another molecule or a rigid body ; in both cases it becomes constrained (Gravity)
4 A constrained molecule can move in any direction only if so requested from one of its neighbours (Fluidity)
5 A constrained molecule becomes free if the space underneath ceases to be occupied either by another molecule or a rigid body (Fluidity)
6 If a free molecule during its fall encounters a constrained molecule, before becoming constrained it will try to occupy its space (Fluidity)
7 A constrained molecule which has received a request for space will try to fulfil this request by attempting to occupy another position in its neighbourhood. If no space is already available it will pass on the request. If no position can be made available it will fail (Fluidity)
8 No position can be occupied above the level of a free molecule which started a request for space (Gravity)

4. Experimental results

The program incorporating these rules gave qualitatively correct simulations of liquid behaviour in a wide variety of situations such as the filling of a container from a tap, leaking from a hole in a container, finding its own level, etc.
By way of illustration Figures 1a to 1e show the filling of a glass by liquid poured from a bottle.
Figures 2a to 2d show a complex of communicating vessels being filled, and susequently leaking through a hole in a wall of a container, into a larger container underneath.
The notion of "qualitatively correct" is not a precise one, being based on the judgment of the observer. However, since essentially we are concerned with modelling perceptions and knowledge of a (competent) observer about physical phenomena, such imprecision needs to be tolerated - for the present, since it is possible that in the future development of naive physics more precise concepts using Hayes' (**Hayes 79a**) notion of quantity space may emerge.

5. Discussion

The model developed so far is essentially a kinematic one,though it encompasses dynamic phenomena like falling under gravity and some of the effects of pressure. It is an open question how far the kinematic approach will carry one when trying to extend the model to deal with a larger range of situations. For instance the present model does not cover splashing, because as yet we have not developed a satisfying way of representing the effects of momentum. An associated open question is the degree of explicitness with which time should be represented.
Analogical representation is applicable not only to modelling physical processes but also to using such models for problem-solving. For instance, a robot waiter that has to serve drinks could have in its planning program such a representation (cf. Figure 1) of glass and bottle, and so find how much to tilt the bottle in the space above the glass.
However, the question of including analogical models in an integrated autonomous problem-solving or reasoning system involves confronting important issues like interpretation, generality and the representation of analogical heuristics, which are discussed in (Cambardella **et al. 86**).

6. References

Bobrow 84
Bobrow, D.G. (Ed.), Artificial Intelligence (Special volume on qualitative reasoning about physical systems) 24 (1984) 1-491.

Feynmann 63
Feynman R., Leighton R. & Sands M. "The Feynman Lectures on Physics", Addison-Wesley Pub. Comp., 1963

Forbus 84
Forbus, "Qualitative Process Theory" in Artificial Intelligence (Special volume on qualitative reasoning about physical systems) 24 , (1984) 85-168.

Gambardella et al. 86
Gambardella L., Gardin F. & Meltzer B. "Analogic representation in Modelling Naive Physics" Submitted for publication

Hayes 73
Hayes, P.J., "The Frame problem and Related Problems in Artificial Intelligence", from Artificial and Human Thinking, 45-59, Jossey-Bass, 1973

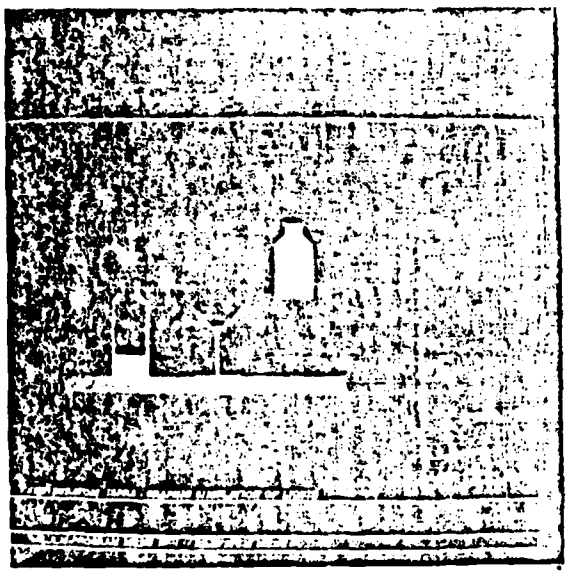
Hayes 79a
Hayes, P.J., "The naive physics manifesto", in: D.Michie (Ed.), Expert Systems in the Microelectronics Age, Edinburgh University Press, Edinburgh, 1979.

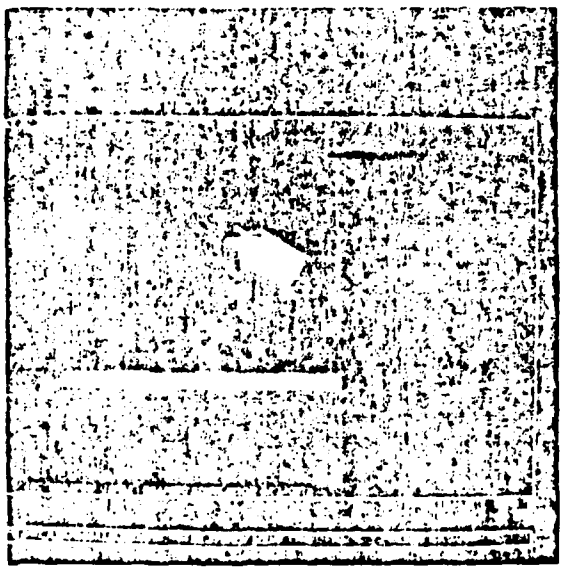
Hayes 79b
Hayes, P.J., "Naive Physics 1 - Ontology for liquids", Centre pour les Etudes Semantiques et Cognitives, Geneva, Switzerland, 1979.

Hillis 85
Hillis, D., "The Connection Machine" ,MIT Press, Cambridge, 1985

Sloman 71
Sloman, A., "Interactions between philosophy and artificial intelligence; the role of intuition and non-logical reasoning in intelligence", Artificial Intelligence 2 (1971) 209-225.

Sloman 84
Sloman, A., "Why we need many knowledge representation formalisms", in: British Computer Society Expert Systems Conference (1984).





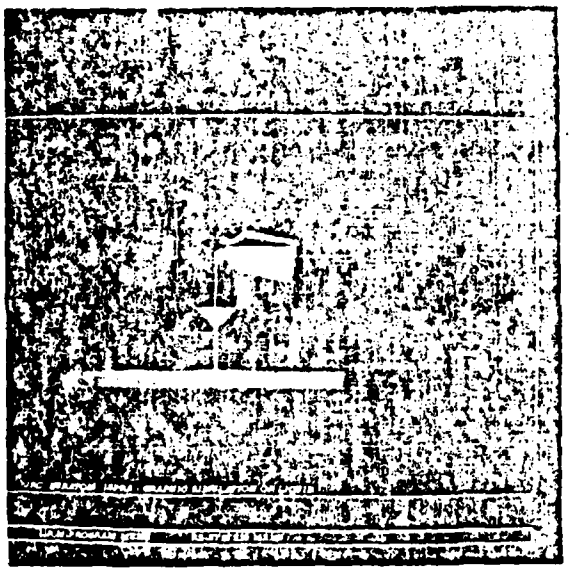
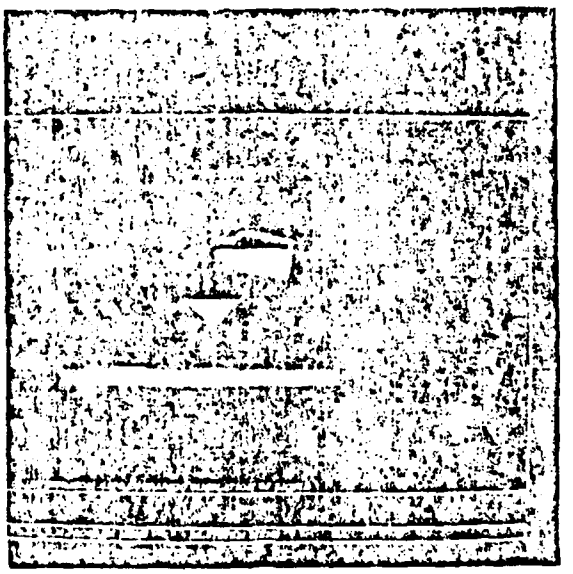
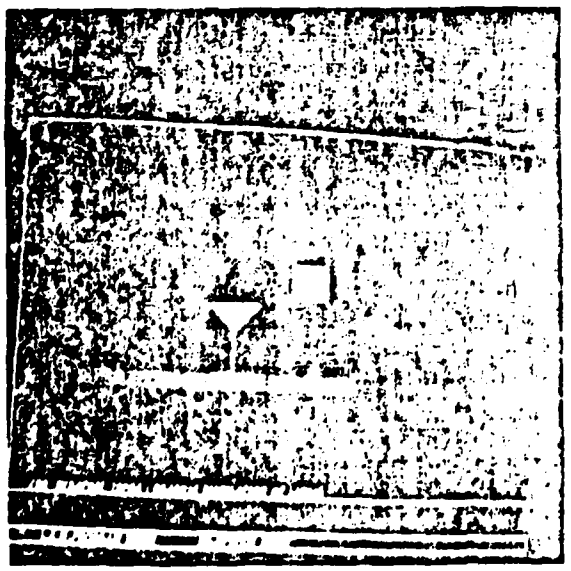


Fig. 1c





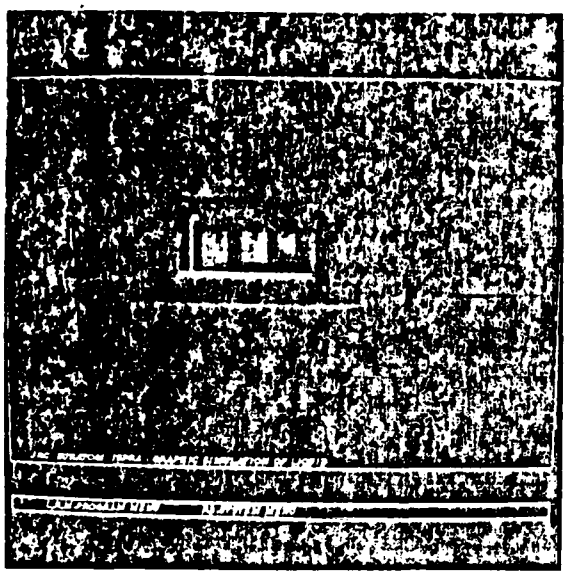
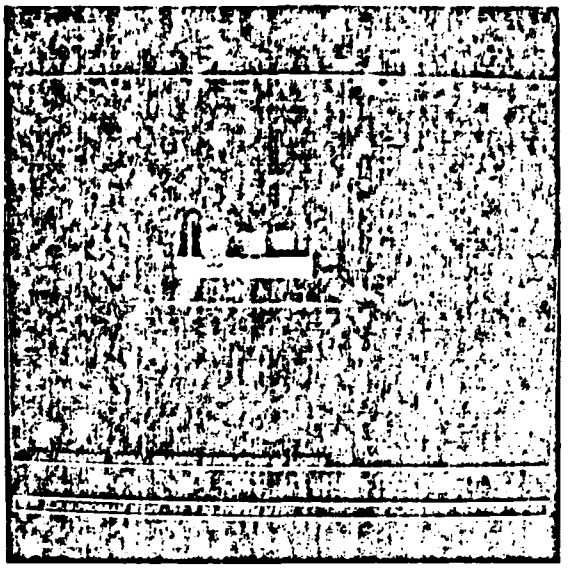
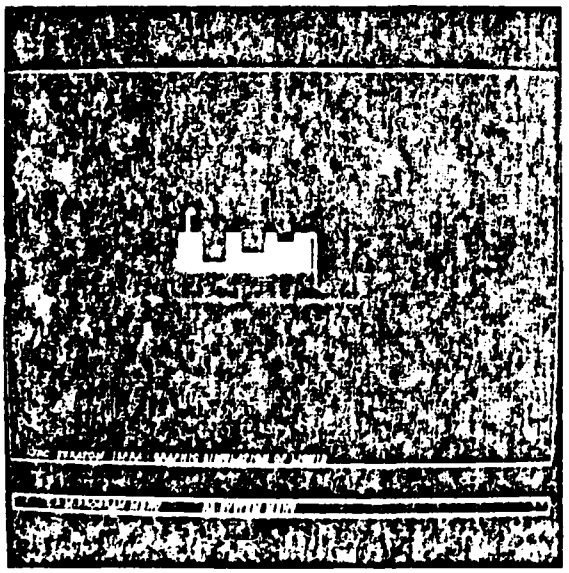
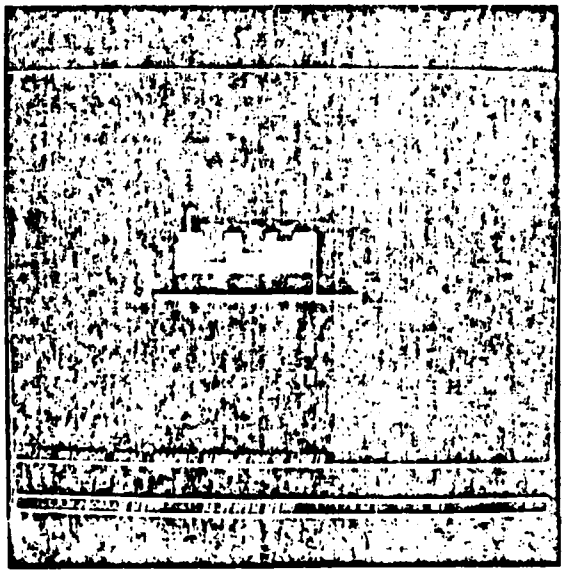


Fig. 2a







KNOWLEDGE-BASED SIMULATION ENVIRONMENTS FOR DYNAMIC SYSTEMS

Pertti Lounamaa
Nokia Corporation, R&D
P.O. Box 780
SF-00101 Helsinki
FINLAND

Edison Tse
Engineering-Economic Systems
Stanford University
Stanford, California 94305
U.S.A.

In the field of mathematical systems analysis there is a long tradition of analyzing dynamic systems via difference and differential equations (Luenberger 1979). This analysis has concentrated on solving for equilibria and on the stability properties of linear systems and of a few special cases of nonlinear systems. Most problems that are of practical interest are in complex ways nonlinear and the properties that one wishes to analyze are often of transient nature. To analyze nonlinear systems and transient phenomena numerical computer simulations are often the only possibility. When the problem has a unique and well known mathematical structure traditional simulation tools are sufficient. In many situations, especially when analyzing poorly understood phenomena, the structure of the model to be analyzed is not predefined. Rather the purpose of the simulation is to experimentally arrive at a good model of a complex system and to perform a wide variety of parametric analyses. For these kind of situations the traditional simulation tools do not provide sufficient support.

Our research has concentraded on developing AI-based simulation tools that form an integrated environment for supporting explorative and experimental analyses of difference equations.

In already completed research (Lounamaa, 1985, Lounamaa and Tse,

1986a) we have developed a Simulation and Expert Environment (SEE) that combines object-oriented programming and rule-based reasoning with various simulation facilities such as repeated execution, noise generation, result storage, plotting and parametric analyses. The features of lisp has been used extensively to provide a powerful and easy to use modeling environment. One may interactively use lisp expressions with the variables refering to columns in result tables to select simulation results for display and analyses. One can trace the simulation in various ways.

SEE has been used to analyze a complex non-linear problem: the adaptive control of a team consisting of learning members within a noisy environment. This class of problems has been analyzed in the literature but only with strong mathematical assumptions. We wanted to start from behaviorally plausible decisionmaking and learning assumptions and thus be able to provide new, behaviorally meaningful, insights to this class of problems. After extensive analyses with a wide variety of alternative formulations and parametric analyses using SEE we have arrived at results that could not have been even anticipated without the help of a tool like SEE (Lounamaa and March, 1986).

The important conclusions with respect to AI-tools are that object-orientation is a powerful unifying framework in the modeling process and that rule-based reasoning had relatively little to contribute in this class of problems. Object orientation naturally leads to what we feel to be good modeling principles. First, one defines that abstract logical model that one wants to analyze by defining the sequence of messages to be sent. Second, one defines the detailed behavior of the objects, that is the responses to the messages. The analysis then consists of changing the behavioral assumptions of the objects while keeping the logical structure unchanged. This modeling

process is managed by defining a new object class for each new behavioral assumption. In this way one can first of all keep track of the various assumptions analyzed and secondly easily switch between the assumptions by changing the simulated objects. Various ways of integrating rule-based reasoning with object-oriented modeling were studied. In the class of problem studied we did, however, not find any uses of (unification based) rule-based reasoning in which its inherent inefficiency would be justified. For instance, behavioral assumptions were more productively expressed as methods of objects.

SEE is a fully working modeling tool prototype that has been used to obtain substantive results. It is, however, implemented in MACLISP and GLISP (Novak 1982) in a timesharing environment and thus is both inefficient and lacks important user interface functionalities. The next generation modeling environment should be implemented in a lisp workstation.

Our current research is focused on defining a new and powerful knowledge-representation formalism for simulating dynamic systems (Lounamaa and Tse, 1986b). This formalism is based on object-orientation and on including semantic knowledge of the class of problems that the representation should be able to describe, namely difference equations of arbitrary complexity. The key mathematical notions that the representation treats semantically correct are state-variables, dependent variables, parameters, difference equations and index sets. The resulting knowledge-representation formalism does not end up talking purely about these mathematical notions but rather a mixture of mathematical notions and AI based notions such object classes, object instances, inheritance, objects "known by" other objects and rules attached to objects. In a complete simulation environment a graphical user interface based on mouse control and

up menus again transforms the way a modeller thinks about and models dynamic systems. This transformation requires further experimental research.

REFERENCES

Novak, Gordon, 1982, GLISP User's Manual, Stanford University, Department of Computer Science.

Luenberger, David G., 1979 , Introduction to Dynamic Systems, Theory, Models, and Applications, John Wiley & Sons, New-York.

Lounamaa, Pertti H., 1985, Models of Multi-Agent Behavior: A Simulation and Expert Environment Approach, Stanford University, Ph.D. thesis.

Lounamaa, Pertti H. and James G. March, 1986, Adaptive Coordination of a Learning Team, submitted to Management Science.

Lounamaa, Pertti H. and Edison Tse, 1986a, The Simulation and Expert Environment, in Janusz Kowalik, ed., Coupling Symbolic and Numerical Computing in Expert Systems. North Holland, Amsterdam.

Lounamaa, Pertti H. and Edison Tse, 1986b, Lisp Based Simulation Modeling, to be presented at the 1st International Conference on Economics and Artificial Intelligence, Aix-en-Provence, France, 2-4 September 1986.

AI AND SIMULATION WORKSHOP:
LIST OF ATTENDEES

Al Adkins
Federal Aviation Administration

R. M. Adler
Arificial Intelligence Applications
The MITRE Corporation
Burlington Road
Bedford, MA 01730

James H. Alexander, Ph.D.
Computer Research Lab
Tektronix, Inc.
P.O. Box 500
Beaverton, OR 97077
503/627-7111

Steven C. Bankes
The Rand Corporation
1700 Main Street
P.O. Box 2138
Santa Monica, CA 90406-2138
213/393-0411

Richard Berenson
Pallaldian Software Inc.
Four Cambridge Center
11th Floor
Cambridge, MA 02142
617/661-7171

Alan Butcher
West Virginia University
Department of Statistics and
Computer Science
Morgan, WV 26506

Jim Calvin
Palladian Software, Inc.
Four Cambridge Center
11th Floor
Cambridge, MA 02142
617/661-7171

Sylvia P. Darnall
General Dynamics
Fort Worth Division
P.O. Box 748
Fort Worth, TX 76101

Matthew DiGeorgi
Advanced Systems Division
Eaton AIL
Walt Whitman Road
Melville, NY 11747

Joe Dilger
182 Wyandotte Drive
San Jose, CA 95123
408/742-8983

Dan Donahue
Texas Instruments Inc.
P.O. Box 225474
Dallas, Texas 75265
214/995-0668

Martin J. Dudziak
Martin Marietta Baltimore Aerospace
Mail Point 600
103 Chesapeake Park Plaza
Baltimore, MD 21220

Steven A. Epstein
Management Analysis Company
P.O. Box 85404
San Diego, CA 92138
619/481-3100

Paul A. Fishwick
Department of Computer and
Information Science
University of Florida
512 Weil Hall
Gainseville, FL 32611

David M. Fram
BBN Laboratories Inc.
10 Moulton Street
Cambridge, MA 02238
617/491-1850

Dr. Mike Frank
Pickard, Lowe and Garrick
2260 University Avenue
Newport Beach, CA 92660

LTC Russell Frug
Defense Advanced Research Projects Agency
ISTO (Air Land Battle Management Project)
1400 Wilson Blvd
Arlington, VA 22209-2308

Prof. Francesco Gardin
Via Rosmini, 7
45100 Rovigo
ITALY
0425-21485

Ina Ghaznavi
Strategic Systems Division
GTE Government Systems Corporation
1 Research Drive
Westborough, MA 01581
617/366-6000

Brian J. Glass
School of Areospace
Georgia Institute of Technology
Atlanta, GA 30332

Sol J. Greenspan
Schlumberger-Doll Research
Old Quarry Road
Ridgefield, Connecticut 06877-4108
203/431-5000

Marc R. Halley
The Analytic Sciences Corporation
8301 Greensboro Drive
Ste. 12000
McLean, VA 22102
703/734-4100

S. Hanagud, Professor
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
404/894-3000

S. L. Hardt
Department of Computer Science
226 Bell Hall Buffalo, NY 14260
716/636-3180

Dr. David H. Helman
201 Guilford
Case Western Reserve University
Cleveland, OH 44106

Bill Huth
Johnson Controls
Milwaukee, Wisconsin

Peter D. Karp
Knowledge Systems Laboratory
Computer Science Department
Stanford University
Stanford, CA 94305

Bill Kessner
Westinghouse

Mark A. Kramer
Massachusetts Institute of Technology
Department of Chemical Engineering
Room #66-401
Cambridge, MA 02139
617/253-6508

Stephen Lakowske
NASA/Ames Research Center
MS 239-19
Moffet Field, CA 94035
415/694-6436

Pauline Langln
UTC

C. N. Lee, Ph.D.
Siemens Corporation Research
and Support Inc.
105 College Raod East
Princeton, NJ 08536
609/734-3346

David C. Littman
Cor[illegible]ition and Programming Project
[illegible]ment of Computer Science
[illegible]versity
[illegible], CT 06520

[illegible]unamaa
[illegible]ctronics
[illegible]780
[illegible]elsinki
[illegible]

Ste[illegible]nald
We[illegible]e

Phil[illegible]ide
Int[illegible]
1[illegible]ino Real West
Moun[illegible]ew, CA 94040

James M. McKinion
US Department of Agriculture
Agricultural Research Service
Crop Science Research Laboratory
P.O. Box 5367
Mississippi State, MS 39762-5367
601/323-2230

Claire D. Meier
Boeing Computer Services
P.O. Box 24346, Mail Stop 7J-64
Seattle, Washington 98124
206/865-3293

DR. R. M. O'Keefe
Virginia Polytechnic Institute
and State University
Department of Computer Science
562 McBryde Hall
Blacksburg, VA 24061
703/961-6931

Tuncer Oren
Computer Science Department
University of Ottawa
Ottawa, Ontario KIN9B4
CANADA

Collen Oresky
ESL
495 Java Drive
P.O. Box 3510
Sunnyvale, CA 94088-3510
408/738-2888

Yuhuan Pan
Worcester Polytechnic Institute
Department of Management
Worcester, MA 01609

Edmund C. Payne
IntelliCorp
1975 El Camino Real West
Mountain View, CA 94040

Robert Prager
McGill University
School of Computer Science
Burnside Hall
805 Sherbooke Street West
Montreal, PQ, CANADA
H3A 2K6
514/392-8275

Dr. Malcolm R. Railey
The BDM Corporation
7915 Jones Branch Drive
McLean, VA 22102-3396
703/821-5000

Douglas J. Roberts
Military Systems
Los Alamos National Laboratory
Los Almos, New Mexico 87545
505/667-4569

Jeff Rothenberg
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406
213/393-0411

N. Sathi
GGI

David Silverman

Joseph Talavage
Purdue University
School of Industrial Engineering
Grissiom Hall
West Lafayette, IN 47907
317/494-5400

Kjell Tangen
Computas Expert Systems
P.O. Box 410
N-1322 Hovik
NORWAY
+47-2-47- 78 05

Ranjeet Uttamsiangh
Electronic Data Systems

Don Waye
Applied High Technology Ltd.
2, Place Yvon Plourde, Suite 205
Charemagne (Quebec)
J5Z 3E8
CANADA
514/582-1461

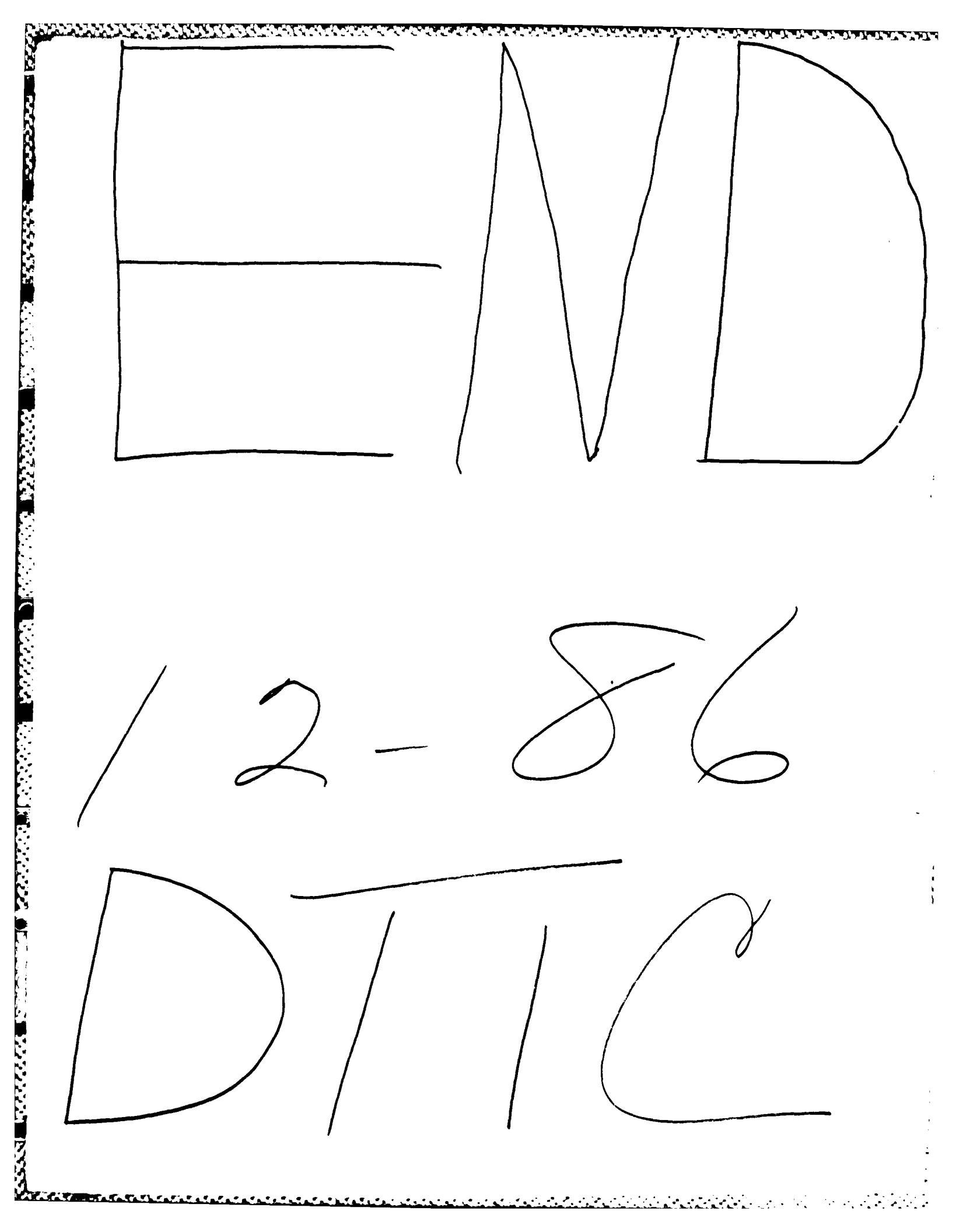
Lawrence E. Widman, M.D.
Division of Cardiology
Department of Medicine
University Hospitals of Cleveland
2065 Adelbert Road
Cleveland, OH 44106

Ben Wise
Thayer School
Dartmouth College
Hanover, N.H. 03755
603/646-2695

Perry A. Zalevsky
Process Control and
Computer Technology Division
Aluminum Company of America
Alcoa Technical Center
Alcoa Center, PA 15069
412/339-6651

Bernard P. Zeigler
Department of Electrical and
Computer Engineering
The University of Arizona
Tucson, Arizona 85721
602/621-2108

Stephen Zvolner
Knowledge Engineering Section
Johnson Controls, Inc.
507 East Michigan Street
P.O. Box 423
Milwaukee, WI 53201
414/274-4000


END
12-86
DTIC